UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA AUTOMATIZAR A MANUTENÇÃO DO CÓDIGO DE PROCEDIMENTOS DE CARGA PARA AMBIENTES DE *BUSINESS INTELLIGENCE*

JULI KELLE GÓIS COSTA

SÃO CRISTÓVÃO/SE
2015

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL
UNIVERSIDADE FEDERAL DE SERGIPE

Costa, Juli Kelle Góis
C837a Uma abordagem para automatizar a manutenção do código de procedimentos de carga para ambientes de *business inteligence* / Juli Kelle Góis Costa ; orientador Methanias Colaço Júnior. – São Cristóvão, 2015.
119 f. : il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Sergipe, 2015.

1. Engenharia de software. 2. Armazenamento de dados. 3. SQL (Linguagem de programação de computador). 4. Inteligência competitiva. 5. Software - Manutenção. I. Colaço Júnior, Methanias, orient. II. Título.

CDU 004.416:005.92

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JULI KELLE GÓIS COSTA

UMA ABORDAGEM PARA AUTOMATIZAR A MANUTENÇÃO DO CÓDIGO DE PROCEDIMENTOS DE CARGA PARA AMBIENTES DE *BUSINESS INTELLIGENCE*

Proposta de Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal do Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Methanias Colaço Júnior

SÃO CRISTÓVÃO/SE
2015

JULI KELLE GÓIS COSTA

UMA ABORDAGEM PARA AUTOMATIZAR A MANUTENÇÃO DO CÓDIGO DE PROCEDIMENTOS DE CARGA PARA AMBIENTES DE *BUSINESS INTELLIGENCE*

Proposta de Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal do Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Prof. Dr. Methanias Colaço Júnior, Presidente
Universidade Federal de Sergipe (UFS)

Prof. Dr. Michel dos Santos Soares
Universidade Federal de Sergipe (UFS)

Prof. Dra. Karin Becker
Universidade Federal do Rio Grande do Sul (UFRGS)

UMA ABORDAGEM PARA AUTOMATIZAR A MANUTENÇÃO DO CÓDIGO DE PROCEDIMENTOS DE CARGA PARA AMBIENTES DE *BUSINESS INTELLIGENCE*

Este exemplar corresponde à redação parcial da Dissertação de Mestrado, sendo o Exame de Defesa da mestranda **JULI KELLE GÓIS COSTA** para ser aprovada pela Banca Examinadora.

São Cristóvão - SE, 27 de AGOSTO de 2015

_____

Prof. Dr. Methanias Colaço Júnior

Orientador

_____

Prof. Dr. Michel dos Santos Soares

Membro

_____

Prof. Dra. Karin Becker

Membro

RESUMO

Grande parte das aplicações de *Business Intelligence* (BI) efetivas depende de um *Data Warehouse* (DW), um repositório histórico de dados projetado para dar suporte a processos de tomada de decisão. Sem um DW eficiente, as organizações tendem a não extrair, em um tempo aceitável, os dados que viabilizam ações estratégicas, táticas e operacionais mais eficazes. Muitos ambientes de BI possuem um processo de Engenharia de Software particular, baseado em dados, para desenvolver programas de Extração, Transformação e Carga (ETL) de dados para o DW. Este trabalho propõe o desenvolvimento e experimentação de uma abordagem de Desenvolvimento Rápido de Aplicações (RAD) para aumentar a eficácia e a eficiência da manutenção de procedimentos de carga SQL, utilizados em processos ETL, avaliando a relação existente entre a sua utilização e a qualidade dos dados que são movidos, gerados e atualizados durante o processo de povoamento de um *Data Warehouse*. Este é um ambiente ímpar que necessita de maior integração e interdisciplinaridade entre as áreas de Engenharia de Software (ES) e Banco de Dados. Foi feita uma avaliação da criação e manutenção automática de procedimentos em extensões da SQL, perfazendo dois experimentos controlados feitos na indústria, para analisar a efetividade de uma ferramenta que encapsula e automatiza parte da abordagem. Os resultados indicaram que a nossa abordagem pode ser usada como método para acelerar e melhorar o desenvolvimento e manutenção de processos ETL.

Palavras-chave: Engenharia de Software Experimental, BI, Manutenção, *Data Warehouse*, ETL.

ABSTRACT

Business Intelligence (BI) relies on Data Warehouse (DW), a historical data repository designed to support the decision making process. Without an effective Data Warehouse, organizations cannot extract the data required for information analysis in time to enable more effective strategic, tactical, and operational insights. This thesis presents an approach and a Rapid Application Development (RAD) tool to increase efficiency and effectiveness of ETL (Extract, Transform and Load) programs creation and maintenance. Experiment evaluation of the approach is carried out in two controlled experiments that carefully evaluated the efficiency and effectiveness of the tool in an industrial setting. The results indicate that our approach can indeed be used as method aimed at improving creation and maintenance of ETL processes.

Keywords: Experimental Software Engineering, maintenance, Data Warehouse, ETL, BI.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

BI	*Business Intelligence*
ETL	*Extraction, Transformation and Load*
DW	*Data Warehouse*
RAD	*Rapid Application Development*
ES	*Engenharia de Software*
SQL	*Structured Query Language*
OLTP	*Online Transaction Processing*
OLAP	*On-line Analytical Processing*
MD	*Modelagem Dimensional*
DER	*Diagrama Entidade Relacionamento*
SGBD's	*Sistema Gerenciador de Banco de Dados*
GUT	*Gravidade, Urgência e Tendência*

SUMÁRIO

CAPÍTULO 1

1. INTRODUÇÃO

Atualmente, para auxiliar as áreas estratégicas das organizações, são criadas grandes bases de dados nomeadas de *Data Warehouses*. Um *Data Warehouse* (DW) representa um banco de dados histórico, separado lógica e fisicamente do ambiente de produção de uma organização, concebido para armazenar dados extraídos essencialmente deste ambiente. Antes de serem armazenados no DW, os dados são selecionados, integrados e organizados, para que possam ser acessados da forma mais eficiente, auxiliando assim o processo de tomada de decisões (COLAÇO, 2004; INMON, 2005; KIMBALL et al., 2008).

Comumente, um DW é a fonte de dados para um conjunto de tecnologias que suportam a análise dos indicadores de desempenho de um negócio, denominado BI (*Business Intelligence*¸ ou Inteligência Aplicada aos Negócios)(COLAÇO, 2004). Neste contexto, grande parte das aplicações de *Business Intelligence* depende de um *Data Warehouse*. Sem um DW efetivo, as organizações tendem a não extrair, em um tempo aceitável, os dados que viabilizam ações estratégicas, táticas e operacionais mais eficazes.

Essa efetividade está diretamente relacionada com a qualidade dos dados que são armazenados no *Data Warehouse*. Desse modo, identificar e solucionar problemas com validade, consistência e integridade dos dados representam preocupações constantes das empresas no processo de utilização de ambientes de *Data Analytics* (RANJIT SINGH; KAWALJEET SINGH, 2010). Os problemas com qualidade de dados podem surgir em várias fases do processo de carga de dados dos Sistemas Transacionais para o DW, conhecido como ETL (*Extract, Transform and Load*), especialmente no estágio de povoamento (KIMBALL et al., 2008). Para efetuar esta carga, é necessário que os dados passem primeiramente por uma área intermediária, conhecida como *Área de Staging* (COLAÇO, 2004; KIMBALL et al., 2008). Essa área intermediária representa a base do *Data Warehouse*, pois é na *Área de Staging* que ocorrem as principais transformações, integrações e tratamento dos dados provenientes do ambiente transacional.

Uma das alternativas usadas para o povoamento de dimensões[1] (perspectivas de análise de um negócio) (SANTOS; BELO, 2011) é a criação de rotinas codificadas, manualmente, por meio de linguagens procedurais que estendem a SQL (KIMBALL et al., 2008; SANTOS; BELO, 2011).

Em paralelo, algumas ferramentas *Open Source* e proprietárias foram desenvolvidas para auxiliar fases do processo ETL. No entanto, muitas empresas optam por codificar várias etapas do processo ETL manualmente. Em geral, o uso dessas ferramentas diminui o controle e, em algum momento, a infraestrutura (sistemas legados) tem que se adequar a processos caixas-preta que não podem ser totalmente analisados (depurados) e customizados. Além disso, pode ser mais cômodo, por questões financeiras, culturais e de curva de aprendizado, permanecer com a codificação manual ao invés de investir em uma ferramenta. Esta questão cultural e financeira foi averiguada em pelo menos 5 clientes da empresa que colaborou com este trabalho. Por fim, empresas que optam por uma plataforma proprietária podem receber a indicação do próprio fornecedor, para o uso de codificação manual em processos ETL (MICROSOFT, 2015).

Apesar dos benefícios com sua utilização, a codificação manual, assim como os erros nas estratégias de implementação do tratamento de dados históricos, são apontadas, dentre outras, como principais causas para a má qualidade de dados gerados em um *Data Warehouse* (RANJIT SINGH; KAWALJEET SINGH, 2010). Desta forma, a utilização de uma abordagem para geração automática na criação e manutenção do código procedural SQL, neste cenário, pode vir a substituir a codificação manual, diminuindo o tempo de codificação, reduzindo o número erros e contribuindo para uma melhor qualidade dos dados.

A manutenção é responsável por um grande custo do ciclo de vida de um software. Durante o processo de manutenção, as mudanças podem ser simples, para corrigir erros de codificação, ou modificações mais extensas, como correção de projeto ou melhorias significativas (PRESSMAN, 2007). Um dos problemas para manutenção de software, segundo (SOMMERVILLE, 2006), é a mobilidade do pessoal de software, pois é provável que a pessoa ou equipe que desenvolveu o sistema legado não esteja por perto e a documentação não esteja atualizada. Diante desse contexto, é importante buscar maneiras para

[1] São tabelas que qualificam os indicadores de desempenho, formadas por atributos de negócio, em sua maioria descritivos, que servem como títulos das colunas em relatórios gerenciais (SANTOS; BELO, 2011).

amenizar esta problemática na manutenção dos diversos códigos-fonte existentes, assim como na codificação de procedimentos SQL.

O objetivo desta dissertação é desenvolver e avaliar uma abordagem para aumentar a eficácia e a eficiência da manutenção, corretiva[2] e aditiva[3], de procedimentos de carga SQL utilizados em processos ETL que alimentam aplicações de *Business Intelligence*, analisando a relação existente entre sua utilização e a qualidade dos dados que são movidos, gerados e atualizados durante o processo de povoamento de um *Data Warehouse*. Pelo exposto anteriormente, nota-se que este é um ambiente ímpar que necessita de maior integração e interdisciplinaridade entre as áreas de Engenharia de Software (ES) e Banco de Dados. Neste tipo de ambiente, os dados fontes já estão claramente definidos e o objetivo principal é documentar como extrair e disponibilizar esses dados dos diversos sistemas legados para os usuários finais (COLAÇO, 2004).

Está sendo proposta uma abordagem, ao invés de um método, pois, segundo o (IEEE, 2010), um método "é um padrão que descreve as características do processo ou procedimento ordenado, utilizados na engenharia de um produto ou de um serviço de realização". Neste contexto, nossa abordagem descreve como empresas poderão utilizar metadados para incrementar o desenvolvimento de procedimentos SQL, baseadas em um modelo conceitual que promove a facilitação deste processo. Foi desenvolvida uma ferramenta para automação de parte do processo, mas sem a definição ou imposição de uma ordem, o que nos fez declinar quanto ao termo "método".

Então, para concepção da abordagem, foi realizada uma revisão, com características sistemáticas, sobre as idiossincrasias dos procedimentos de povoamento de dimensões em um ambiente de *Data Warehouse*. A partir do levantamento dessas especificidades, foi idealizado um esquema de dados que pudesse capturar a semântica necessária para geração de rotinas de forma automática. Em seguida, foi construído um *framework*[4] e uma ferramenta, nomeada FGCod, a qual automatiza parte da nossa abordagem, gerando código em extensões SQL.

Nossa ferramenta pretende fornecer características de Desenvolvimento Rápido de Aplicações (RAD) para seus usuários programadores de processos ETL, gerando

[2]Tipo de manutenção a qual faz reparação de erros de codificação (SOMMERVILLE, 2006).
[3]Tipo de manutenção para adicionar ou modificar funcionalidades. Ela é necessária quando os requisitos de sistema mudam em resposta à mudança organizacional ou de negócio (SOMMERVILLE, 2006).
[4] Um projeto de subsistema composto por uma coleção de classes abstratas e concretas e interfaces entre elas. *Frameworks* são muitas vezes instanciação de uma série de padrões (IEEE, 2010).

procedimentos para o SQL Server ou Oracle. Estes dois Sistemas Gerenciadores de Banco de Dados (SGBDs) são duas extensões implementadas e liberadas para uso, ou seja, é possível estender o *framework* para geração de procedimentos para outros bancos. Os resultados indicaram que a FGCod pode aumentar a efetividade do processo de criação e manutenção destes procedimentos.

1.1 Problemática e Hipótese

Para este projeto, o problema maior em questão é averiguar a possibilidade de maior eficiência e eficácia da codificação em procedimentos utilizando a abordagem que automatiza o processo. Para todo o texto restante, leia-se manutenção como sendo manutenção corretiva e perfectiva.

Dessa forma, devemos nos atentar aos seguintes questionamentos sobre essa pesquisa:

a. Questão 1: a codificação automatizada pode diminuir o tempo dos programadores no processo de povoamento de um DW?
b. Questão 2: a codificação automatizada pode reduzir ou eliminar erros na codificação de procedimentos e nos dados povoados em um DW?

Logo, temos duas Hipóteses.

Hipótese 1

Pretendemos negar a hipótese H0:

- Hipótese nula H0: A codificação automática e manual têm a mesma eficiência.
- Hipótese alternativa H1: A codificação automática é mais eficiente que a codificação manual.

Hipótese 2

Pretendemos negar a hipótese H0:

- Hipótese nula H0: A codificação automática e manual têm a mesma eficácia.
- Hipótese alternativa H1: A codificação automática é mais eficaz que a codificação manual.

1.2 Objetivos da Dissertação

O objetivo principal desta dissertação é desenvolver e avaliar uma abordagem para aumentar a eficácia e a eficiência da manutenção, corretiva[5] e aditiva[6], de procedimentos de carga SQL utilizados em processos ETL que alimentam aplicações de *Business Intelligence*, analisando a relação existente entre sua utilização e a qualidade dos dados que são movidos, gerados e atualizados durante o processo de povoamento de um *Data Warehouse*.

Objetivos Específicos

Para alcançar o objetivo primário anterior, alguns objetivos específicos têm que ser atendidos, a saber:

- Definir Abordagem com base em metadados de carga;
- Desenvolver *framework* e estendê-lo para criação de ferramenta que automatiza a criação de procedimentos SQL;
- Desenvolver módulo da ferramenta para manutenção de código SQL em Ambiente DW;
- Realizar experimentos para comparar o desenvolvimento e manutenção de código SQL de forma manual e automática, utilizando módulos da ferramenta;
- Submeter artigos com os resultados dos experimentos.

1.3 Metodologia

A metodologia utilizada para o trabalho em questão consiste, em termos de classificação, em uma pesquisa exploratória (SEVERINO, 2008), prática, estudo de caso, de laboratório e experimental. Exploratória, porque serão explorados/estudados materiais acessíveis ao público em geral, como livros, artigos e ferramentas. Prático, pois será implementada uma ferramenta para codificação automática em ambientes de apoio à decisão.

Também será classificada como estudo de caso, laboratório e experimental, devido ao estudo de caso, a ser realizado com o *framework*, e à execução de experimentos controlados, nos quais serão selecionados grupos de pessoas para utilização da ferramenta desenvolvida.

[5]Tipo de manutenção a qual faz reparação de erros de codificação (SOMMERVILLE, 2006).
[6]Tipo de manutenção para adicionar ou modificar funcionalidades. Ela é necessária quando os requisitos de sistema mudam em resposta à mudança organizacional ou de negócio (SOMMERVILLE, 2006).

Quando falamos em experimento, estamos nos referindo a um tipo de pesquisa científica na qual o pesquisador observa a variação de variáveis dependentes por meio da manipulação e controle de uma ou mais variáveis independentes (KERLINGER; LEE, 1973). As variáveis Independentes são aquelas que são manipuladas e as Dependentes são o efeito, consequência e o resultado observado da influência das variáveis Independentes. Na Engenharia de Software, então, a Experimentação refere-se à combinação de fatos, suposições, hipóteses, especulações e crenças que abundam na construção de software (JURISTO; MORENO, 2001), sendo manipulados assim, artefatos pertencentes a esta construção, para obtenção de alguma resposta.

Diante da classificação da metodologia, primeiramente será realizada uma revisão de literatura, revendo conteúdos necessários para definição da abordagem, os mesmos serão descritos no Capítulo 2. Após esta revisão, serão definidos os metadados necessários para capturar a semântica necessária para o povoamento de cada subsistema. Definidos os metadados, será realizado o desenvolvimento do *framework*, o qual, posteriormente, será estendido pelos módulos da ferramenta, a serem desenvolvidos, de forma iterativa e incremental.

Após o desenvolvimento dos módulos, serão executados os experimentos a fim de acatar ou refutar as hipóteses do trabalho. Abaixo seguem as etapas que serão realizadas para a execução dos experimentos.

1. *Criação do ambiente de Data Warehouse:* nessa fase será definido e criado o ambiente de DW que contenha o esquema dimensional e a Área de *Staging*. Estes servirão como base central para o início de todo o experimento.
2. *Definição dos Casos de Uso para as rotinas de povoamento:* serão definidas especificações para povoamento de dados a serem seguidas pelos programadores que farão parte do experimento.
3. *Revisão de conceitos básicos sobre rotinas de povoamento para o grupo de programadores:* antes de iniciar qualquer etapa junto com os colaboradores, haverá uma revisão sobre as rotinas de povoamento para ambientes de DW.
4. *Treinamento da ferramenta de geração automática de código:* um treinamento será dado para os programadores, a fim de que eles possam se familiarizar com a ferramenta.

5. *Solicitação de criação dos Casos de Uso de forma manual:* os programadores deverão programar as rotinas de povoamento manualmente. Isto servirá para fazer a comparação com as rotinas geradas pela ferramenta.
6. *Solicitação de criação dos Casos de Uso de forma automática:* nesta etapa, os colaboradores irão utilizar a ferramenta para criar as rotinas de povoamento e verificar se o código consegue atingir o propósito de povoar o esquema dimensional corretamente.
7. *Solicitação de manutenção nos Casos de Uso de forma manual:* serão fornecidos procedimentos SQL e os programadores deverão acrescentar, retirar ou corrigir dados do mesmo, de forma manual.
8. *Solicitação de manutenção nos Casos de Uso de forma automática:* serão fornecidos procedimentos SQL e os programadores deverão acrescentar, retirar ou corrigir dados do mesmo, de forma automática, utilizando a ferramenta.

Ao término dos experimentos, serão analisadas as métricas que servirão de base para avaliar as hipóteses. Neste momento, serão analisadas duas métricas: *a) tempo de desenvolvimento* – verificar o tempo gasto para a codificação de cada procedimento; *b) número de erros de codificação* – número de erros de codificação encontrados na fase de codificação dos procedimentos.

1.4 Organização da Proposta

O texto desta proposta de dissertação estará organizado em 7 capítulos que fornecerão toda a base conceitual e experimental para o entendimento completo da proposta. Os tópicos a seguir descrevem o conteúdo de cada um destes capítulos.

- O Capítulo 1 apresentou esta Introdução;
- O Capítulo 2 apresenta o Referencial teórico referente à Arquitetura de um *Data Warehouse* e o processo de criação de um *framework.*
- No Capítulo 3, é apresentada a Abordagem proposta neste trabalho, inicialmente sendo mostrada uma visão com modelo conceitual e, em seguida, todo o processo de criação, implementação e estudo de caso realizado com o *framework.* Por fim, é feita a apresentação da ferramenta FGcod, com o módulo de criação automática dos procedimentos SQL.

- O Capítulo 4 apresenta o Planejamento, Operação e Resultados do Experimento I, realizado na indústria com o módulo da ferramenta para criação de procedimentos.
- No capítulo 5, está descrito o planejamento para o Experimento II, o qual tratará da automatização da manutenção dos códigos SQL para carga de dados.
- O capítulo 6 descreve os trabalhos relacionados à proposta em questão.
- E, finalmente, o capítulo 7, apresenta as conclusões da Dissertação.

CAPÍTULO 2

2. REFERENCIAL TEÓRICO

Este capítulo descreve os conceitos necessários para embasamento de todo o trabalho em questão. Na seção 2.1, será conceituado o termo *Business Intelligence.* Na seção 2.2, está descrita a arquitetura de um *Data Warehouse*. Já a seção 2.3 trata sobre Manutenção de Software. A Seção 2.4 explana sobre qualidade de dados, e, por fim, a seção 2.5 introduz a Matriz GUT.

2.1 Business Intelligence (BI)

Business Intelligence, ou simplesmente *BI*, significa Inteligência Aplicada aos Negócios. O termo está relacionado com todos os aspectos que envolvem sistemas para transformação de dados em informações relevantes. De forma resumida, é todo processo que envolve coleta de dados de diversas fontes, organização, análise e compartilhamento com os executivos interessados da empresa. Estes transformarão estas informações relevantes em decisões importantes para a empresa.

Pode-se dizer que *BI* "é um conjunto de tecnologias que permitem cruzamento de informações e suportam a análise dos indicadores de desempenho de um negócio" (COLAÇO, 2004). Logo, as ferramentas de apoio à decisão que fazem inferência com armazém de dados histórico, como um DW, são também chamadas de ferramentas de *BI.*

Ralph Kimball, prof. PhD é um dos precursores dos conceitos de armazém de dados e sistemas de apoio à decisão. Seus estudos influenciam até hoje a maioria dos projetos de Inteligência de Negócios das empresas. Sua metodologia é conhecida como modelagem dimensional, a qual ensina como tornar sistemas transacionais em sistemas orientados ao mundo empresarial, permitindo uma maior organização, melhor compreensão e rapidez (KIMBALL; ROSS, 2002). Com o tempo o *Business analytical (*análise de negócios) foi introduzido para representar o componente analítico fundamental no BI (DAVENPORT,

2006). Desde então, muitos utilizam o termo *Business Intelligence and Analytics* (BI & A) como um termo unificado para tratar grandes análise de dados.

O BI tem se tornado uma crescente área, despertando cada vez mais o interesse tanto do mercado como da classe acadêmica. Com a evolução da Web e da proliferação de dados no meio mobile, a área de BI também tende a evoluir.

Como uma abordagem centrada em dados, BI & A tem as suas raízes no campo da gestão de banco de dados de longa data. Ele depende muito de várias coletas de dados, extração e tecnologias de análise (CHAUDHURI; DAYAL; NARASAYYA, 2011). As tecnologias e aplicações de BI & A atualmente adotadas na indústria podem ser consideradas como BI & A 1.0, onde os dados são mais estruturados, recolhidos por empresas por meio de vários sistemas legados, e muitas vezes armazenados em sistemas de gerenciamento de banco de dados relacionais (CHEN; STOREY, 2012).

Web *Intelligence*, web *analytics*, e os conteúdos gerados por usuários, coletados por meio da Web 2.0 (DOAN; RAMAKRISHNAN; HALEVY, 2011) marcou o início de uma nova era de BI & A 2.0, em meados da década de 2000, centrado em análise de texto e web para conteúdos web não estruturados. Uma imensa quantidade de informações de clientes podem ser recolhidas a partir da web e visualizadas a partir de várias técnicas de mineração de texto e web. Ao analisar registros de dados podem ser revelados padrões de compras, por exemplo.

Acompanhando a Web 3.0 surge também o BI 3.0, com a era dos dispositivos móveis. *Business Intelligence* 3.0 pode ser acessado por usuários de negócios de todos os níveis, a partir de contadores e executivos de nível C para supervisores do armazém, gerentes de compras, vendas e marketing pessoal. Em vez de gerar relatórios mensal, semanal ou diariamente, BI 3.0 é capaz de gerar relatórios em tempo real.

O Quadro 1 mostra as principais características de acordo com a evolução do BI.

Quadro 1: Evolução BI&A: principais características (CHEN; STOREY, 2012).

	Principais características
BI&A 1.0	Com base em SGBD - conteúdo estruturado • Banco de dados relacionais e data warehousing • ETL e OLAP • *Dashboards e scorecards* • A mineração de dados e análise estatística
BI&A 2.0	Com base na Web, o conteúdo não-estruturado • Recuperação e extração da informação • Parecer de mineração • Pergunta de resposta • *Web analytics* e *web intelligence* • Análise de mídia social • Análise de redes sociais • Análise espacial-temporal
BI&A 3.0	Conteúdo móvel - baseado em sensores • análise de reconhecimento de localização • análise centrada na Pessoa • análise relevante ao contexto • visualização Móvel & IHC (interação Homem-Computador)

Mesmo com evolução do BI, aplicações podem continuar acessando dados de um DW corporativo, o qual precisa e deve ser carregado. Esta carga pode continuar sendo feita com codificação manual, mantendo-se a mesma situação aludida ao longo deste trabalho. Em outras palavras, independente do tipo de disponibilidade do BI (on-line ou não) e de onde os

dados vêm ou para onde vão, como por exemplo, para aplicações móveis, os desafios com cargas para um DW corporativo continuam os mesmos.

2.2 Arquitetura de um *Data Warehouse*

Kimball define um *Data Warehouse* (DW) como "uma cópia das transações de dados especificamente estruturada para consultas e análises" (KIMBALL et al., 2008). Inmon, a quem é creditado o termo DW, dá uma definição mais detalhada. Ele diz que o DW é uma coleção de dados orientada por assunto, integrada, não-volátil, variante no tempo, que dá apoio às decisões da administração (INMON, 2005). Mas, as duas definições enfatizam a análise de dados e suporte à tomada de decisões gerencias, utilizando o histórico de dados presente em um DW.

Na Figura 1 é ilustrada a arquitetura genérica de um ambiente de *Data Warehouse*. Nela, podemos ver os principais componentes da arquitetura: o ambiente OLTP, a Área de *Staging* e o *Data Warehouse*.

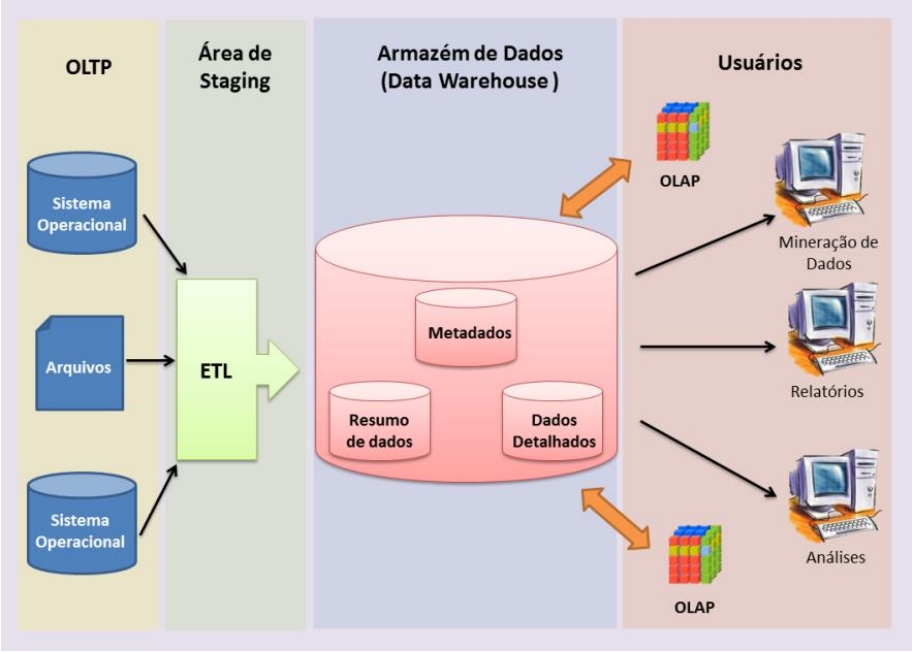

Figura 1: Arquitetura de um DW. Adaptada de (KIMBALL; ROSS, 2002).

O ambiente que representa os sistemas envolvidos nos processos diários das organizações é conhecido como **OLTP** (Online Transaction Processing, em português, Processamento de Transações em Tempo Real). Esses sistemas representam as fontes de informações para o *Data Warehouse*. Antes de serem carregados para as estruturas finais e disponibilizados para os tomadores de decisões, os dados operacionais passam por um

processo conhecido como ETL (Acrônimo de Extract-Transform-Load), responsável pelas principais transformações, integrações e tratamento dos dados provenientes do ambiente transacional. Esse processo ocorre em uma área intermediária conhecida como **Área de *Staging*** (COLAÇO, 2004; KIMBALL et al., 2008). Essa área intermediária representa a base do DW. A fase final do processo de ETL consiste na carga ou povoamento dos esquemas de dados do *Data Warehouse*.

Depois que os dados já estiverem carregados no DW, estes serão submetidos a análises e manipulações por meio de ferramentas OLAP (*On-line Analytical Processing*). Estas ferramentas são utilizadas por gestores em qualquer nível da organização, permitindo-lhes análises comparativas que facilitam a tomada de decisões, pois, a partir destas, é possível realizar mineração de dados, análise estratégica e elaboração de relatórios.

2.2.1 Modelagem Dimensional

A Modelagem Dimensional (MD) é a técnica de projeto lógico de banco de dados mais recomendada para aplicações de suporte à decisão. Os esquemas criados com essa técnica contêm a mesma informação que os esquemas derivados de um DER (Diagrama Entidade Relacionamento), mas, agrupam os dados em uma forma que melhora a clareza do usuário e desempenho das consultas por meio da diminuição do grau de normalização. Os esquemas de dados criados com a Modelagem Dimensional recebem o nome de Esquema Estrela (KIMBALL et al., 2008).

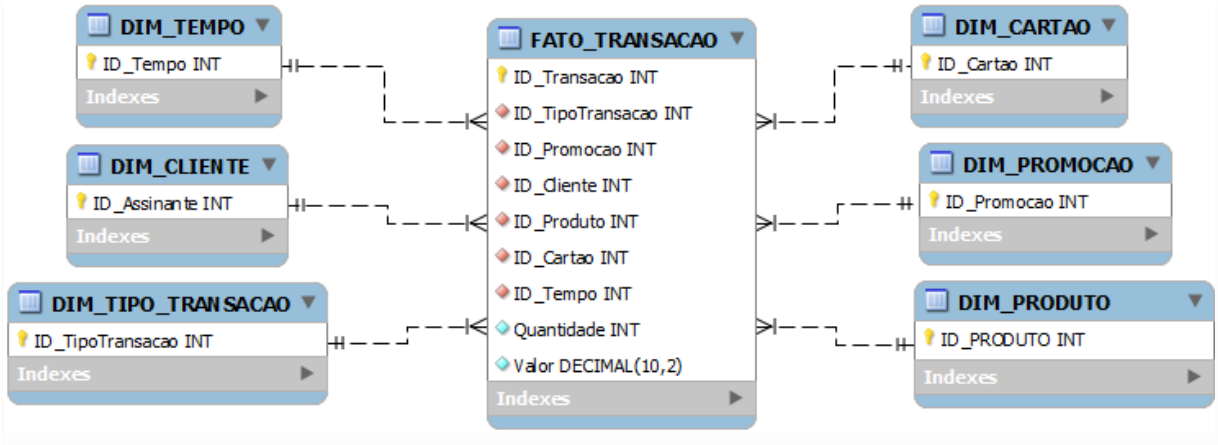

Figura 2: Exemplo de um Esquema Estrela Produzido através da Modelagem Dimensional.

Na Figura 2 é ilustrado um exemplo de um esquema estrela. Um esquema estrela consiste de uma tabela principal, denominada tabela de fatos, e de tabelas auxiliares conhecidas como dimensões. As tabelas de fatos são responsáveis por registrar as transações, ocorrências ou fatos de um processo de negócio e as dimensões representam os elementos que irão contextualizar os fatos registrados. É na fase de povoamento do processo ETL que as tabelas de fatos e dimensões são povoadas. As rotinas responsáveis por essa fase são as principais responsáveis pelo tratamento e armazenamento correto do histórico de dados dentro de um ambiente de suporte à decisão. Nas próximas seções, são apresentadas as principais características dos fatos e dimensões.

2.2.2 Tabela de Fatos

São nas tabelas de fatos que são guardadas as medidas numéricas mais importantes do processo de negócio sendo modelado. Elas são as principais tabelas em um esquema estrela (KIMBALL et al., 2008). As tabelas de fato armazenam grande quantidade de dados históricos em função do tempo, obtidos a partir da intersecção de todas as dimensões da estrela (COLAÇO, 2004). As dimensões são ligadas aos fatos através das suas *Surrogate Keys* (chaves primárias).

As tabelas de fatos possuem uma propriedade conhecida como grão ou granularidade. A granularidade de uma tabela de fatos representa o nível de detalhe associado com as medidas armazenadas. Ao centro da Figura 2 é ilustrado um exemplo de tabela fatos, FATO_TRANSACAO, cuja granularidade é a venda de um produto a um cliente em determinado dia. Uma tabela de fatos é composta por dois tipos de atributos: a) Os atributos que representam as ligações com as dimensões; b) Os atributos que representam as medidas do negócio. Na tabela de fatos da Figura 2, por exemplo, temos os seguintes atributos que representam as ligações com as dimensões: Id_TipoTransacao, Id_Promocao, Id_Tempo, Id_Cliente, Id_Produto, Id_Cartao. Os atributos Quantidade e Valor representam as medidas para o fato registrado.

2.2.3 Dimensões

As dimensões são a principal fonte de informações e restrições no processo de análise. Por isso, seus atributos devem ser valores descritivos (IMHOFF CLAUDIA, GALEMMO

NICHOLAS, 2003). As dimensões representam tabelas cujos atributos são utilizados para contextualizar, restringir e agrupar consultas direcionadas para tabelas de fatos dentro de um esquema dimensional.

As tabelas de dimensões são compostas basicamente por colunas, nas quais são contidos elementos textuais que descrevem o negócio, e uma chave primária, na forma de uma *Surrogate Key*. Outras colunas podem surgir como padrão para compor as tabelas de dimensões, a depender da abordagem para tratamento de histórico. A Figura 3 ilustra um exemplo de uma dimensão. Nela, podemos identificar o atributo que representa a chave primária, Id_cliente, e os atributos que descrevem as características do cliente modelado.

A Figura 3 representa um exemplo de tabela dimensão de cliente.

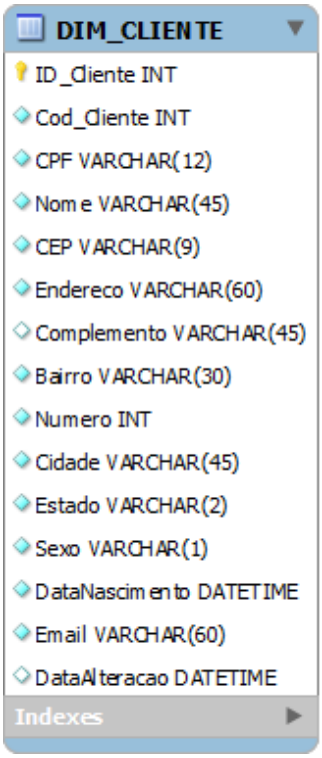

Figura 3: Exemplo de Tabela Dimensão (Tabela Dimensão Cliente).

Tratamento de Histórico em Dimensões

As mudanças dos valores dos atributos das dimensões podem ser relevantes para a análise de dados e tomada de decisões. A maneira como essas mudanças são tratadas pelo esquema de dados do *Data Warehouse* determina o tratamento de histórico que será dado para cada atributo da dimensão. Os atributos de uma dimensão podem ser classificados, em função do tratamento de histórico, nos seguintes tipos: Tipo 0, Tipo 1, Tipo 2 e Tipo 3 (SANTOS;

BELO, 2011). Embora a literatura (KIMBALL; ROSS, 2002) apresente outros tipos de atributos de dimensões, Tipo 4 e Tipo 6, esses últimos são apenas variações, envolvendo os três primeiros tipos.

TIPO 0

Quando um atributo tem seu comportamento, em relação ao histórico, classificado como Tipo 0 significa dizer que o atributo armazena sempre o valor original. Em outras palavras, será descartada qualquer mudança que possa ocorrer com o atributo (SANTOS; BELO, 2011).

TIPO 1

Não há armazenamento de histórico para atributos da dimensão do Tipo 1. O comportamento do atributo, para este tipo, sobrescreve o valor antigo de uma linha da dimensão pelo valor atual proveniente da Área de *Staging*. Dessa forma, o atributo sempre irá refletir o valor mais recente proveniente do ambiente operacional. Ao serem identificados novos valores, que ainda não constam na dimensão, o procedimento cria um novo registro no DW.

TIPO 2

O Tipo 2, ao contrário do comportamento do tipo anterior, é utilizado quando há a necessidade de armazenar histórico dos atributos. O tratamento de histórico é realizado através da criação de novos registros. É gerado um novo valor do identificador da dimensão (Surrogate Key) quando há alteração em atributos que necessitam armazenar todo o seu histórico.

No esquema da dimensão que possui atributos do Tipo 2 surgem novos atributos que não existem na dimensão que contempla somente atributos do Tipo 1. Esses novos atributos são utilizados para o tratamento dado ao histórico dos atributos do tipo 2. São eles: a data inicial, que representa a data em que o registro foi gravado; a data final, que representa a data em que o registro deixou de ser corrente; e por fim, o atributo que identifica se é um registro atual ou não.

TIPO 3

O tratamento de histórico dos atributos do Tipo 3 é utilizado quando há uma mudança e existe a necessidade de manter o histórico sem a criação de um novo registro. Essa característica é alcançada através do armazenamento do histórico em vários atributos de um

mesmo registro de dados. Logo, a partir da necessidade, podem ser criadas quantas colunas forem desejadas para a dimensão.

Esta pesquisa lida com o tratamento de histórico dos atributos dos tipos 1, 2 e 3, sendo que a abordagem em questão gerará procedimentos para carga das dimensões, dando suporte aos três tipos de históricos para atributos. Embora atenda aos três tipos, os experimentos se darão em torno dos tipos 1 e 2, por serem os mais utilizados nas empresas.

2.3 Manutenção de Software

Um sistema de software precisa ser alterado constantemente, devido a mudanças nas necessidades dos utilizadores, a tecnologia e as condições do negócio. Assim, os custos que aparecem mesmo após o desenvolvimento de software, devido às alterações, devem ser incorporadas. Para os sistemas de software que têm uma longa vida útil, esses custos podem ser 3 ou 4 vezes o custo de desenvolvimento (SOMMERVILLE, 2001).

As mudanças mais comuns, além da correção de erros que os sistemas sofrem são: migração para novas plataformas, extensão de funcionalidade e mudança de hardware ou S.O.. Em geral essas mudanças são realizadas sem que haja a preocupação com a arquitetura geral do sistema, produzindo estruturas mal projetadas, documentação desatualizada, lógica e codificação ruins, sendo esses os focos que dificultam a manutenção em um sistema (OSBORNE; CHIKOFDRY, 1990).

A manutenção de software é mais do que apenas consertar erros, de acordo com (SOMMERVILLE, 2006), apenas 20% de todo trabalho de manutenção é gasto "consertando erros". Os 80% restantes são gastos adaptando sistemas existentes a modificações no seu ambiente externo, fazendo melhorias solicitadas por usuários e submetendo uma aplicação a reengenharia[7], para o futuro.

A Manutenção de Software pode ser definida pela identificação de quatro atividades diferentes: manutenção adaptativa (novos formatos, *refactoring*) corretiva (corrigir erros), perfectiva (aditiva) ou de melhoria (adicionar novas funcionalidades) e preventiva ou reengenharia (SOMMERVILLE, 2006).

[7] A reengenharia atua, analisando o sistema procurando reestruturá-lo ou reconstruí-lo com maior qualidade (PRESSMAN, 2007).

Dentre estes tipos, neste projeto, serão trabalhadas a manutenção corretiva, quando houver necessidade de fazer alguma correção na codificação de um Procedimento SQL, como por exemplo, a correção de alguma lógica do procedimento que estava fazendo com que dados não fossem carregados de maneira correta, bem como a manutenção aditiva, quando houver modificação de alguma regra de negócio, como por exemplo, a adição ou exclusão de um atributo, alteração de um tipo de histórico para determinado atributo.

2.4 Qualidade de Dados

A definição de qualidade de dados trata-se de dados que estão ausentes, incorretos, inválidos ou até mesmo duplicados, em algum contexto, o que leva à incapacidade destes dados atingirem um valor de qualidade. Uma definição mais ampla é que a qualidade dos dados é alcançada quando a organização utiliza dados que são abrangentes, compreensíveis, consistentes, relevantes e oportunos (IDRIS; AHMAD, 2011). Compreender as principais dimensões de qualidade de dados é o primeiro passo para a melhoria de qualidade dos dados. Para serem processados e interpretados de forma eficaz e eficiente, os dados têm que satisfazer um conjunto de critérios de qualidade. Os dados que satisfazem esses critérios de qualidade são ditos serem de alta qualidade. Dimensões da qualidade dos dados tipicamente incluem precisão, confiabilidade, relevância, consistência, precisão, pontualidade, finura, compreensibilidade, concisão e utilidade (IDRIS; AHMAD, 2011).

Em um estudo realizado nos Estados Unidos, estima-se que o custo de dados de baixa qualidade, varia entre 8 - 12%, do custo operacional das empresas (TAYI; BALLOW, 1998). Isto mostra a percentagem de importância da qualidade dos dados em processos organizacionais. A qualidade dos dados e precisão são características importantes em análise de dados, pois, má qualidade ou dados imprecisos pode levar a decisões erradas em qualquer cenário (RIBEIRO; GOLDSCHMIDT; CAVALCANTI, 2011).

Muitos problemas na qualidade de dados são identificados em empresas que codificam manualmente suas rotinas de povoamento para DW, gerando problemas de qualidade pela falta de uma padronização nessa codificação. Podem ser citados alguns erros encontrados em empresas colaboradoras deste projeto, os quais ocasionam má qualidade nos dados, tais como: eliminação e inserção de registros já encontrados na dimensão; falta de atualização para alguns atributos; não atualização de dados e duplicação de dados.

2.5 Matriz GUT

Segundo (MARSHALL et al., 2011) matriz GUT é a representação de problemas ou riscos potenciais, através de quantificações que buscam estabelecer prioridades para abordá-los, visando diminuir os impactos.

Esta metodologia propicia a uma empresa ou situação específica a definir suas estratégias e políticas a média e longo prazo, priorizando as mais importantes levando em consideração alguns parâmetros. Ela tem grande utilidade para a fixação de prioridades na eliminação de problemas, consistindo em separar e priorizar os problemas para fins de análise e posterior solução.

Uma Matriz GUT é feita a partir de parâmetros baseados na gravidade(G), urgência (U) e tendência (T) que são tomados para estabelecer prioridades na eliminação de problemas, orientando assim as decisões mais complexas (GONÇALVES, 2011). Cada um destes aspectos possui um impacto que deve ser analisado e atribuído de acordo com os problemas.

Estes aspectos são definidos da seguinte forma (ABRANTES, 2009):

- Gravidade – G. A atividade é analisada e faz-se a seguinte pergunta: quais as consequências caso a atividade não seja realizada prioritariamente?
- Urgência – U. Faz-se a seguinte pergunta: com que urgência a atividade deve ser realizada?
- Tendência – T. Analisa qual a tendência da não execução imediata da atividade. Pode-se pensar na seguinte pergunta: qual a situação pela não execução imediata da atividade?

Habitualmente atribui-se valores entre 1 e 5, a cada uma das dimensões (G.U.T), correspondendo o 5 à maior intensidade e o 1 à menor. Os valores obtidos para o G, U e T, serão multiplicados a fim de se obter um valor para cada problema ou fator de risco estudado (TRISTÃO, 2011). Os problemas ou fatores de risco que obtiverem maior pontuação serão tratados prioritariamente (MARSHALL et al., 2011).

O Quadro 2 mostra um exemplo básico da estrutura de uma matriz GUT.

Quadro 2: Exemplo básico de matriz GUT (ABRANTES, 2009).

Valor	**Gravidade** (Consequências se nada for feito)	**Urgência** (Prazo para tomada de decisão)	**Tendência** (Proporção de problema no futuro)	**Valor Total G x U x T**
5	Os prejuízos ou dificuldades são extremamente graves	É necessária uma ação imediata	Se nada for feito, o agravamento da situação é imediata	5x5x5 = 125
4	Muito grave	Com alguma urgência	Vai piorar no curto prazo	4x4x4 = 64
3	Graves	O mais cedo possível	Vai piorar no médio prazo	3x3x3 = 27
2	Pouco grave	Pode esperar um pouco	Vai piorar no longo prazo	2x2x2 = 8
1	Sem gravidade	Não tem pressa	Não vai piorar (e pode até melhorar)	1x1x1 = 1

A matriz GUT foi utilizada, neste trabalho, como forma de medir o grau de criticidade dos erros encontrados nas cargas de dados geradas pelos procedimentos criados, de forma manual e automática.

CAPÍTULO 3

3. UMA ABORDAGEM PARA CODIFICAÇÃO AUTOMÁTICA DE CÓDIGO SQL

3.1 Visão Geral da Abordagem

Como comentado anteriormente, muitas empresas realizam as cargas de dados da Área de *Staging* para o DW utilizando procedimentos SQL. Este método pode gerar alguns problemas, seja na qualidade dos dados que vão para DW, ou até mesmo, no tempo gasto para criar e manter estes códigos de forma manual. Para estas empresas, é interessante uma abordagem que mitigue esses riscos provenientes do processo de geração do código.

Diante desse contexto, foi realizada uma pesquisa, obtendo informações a partir de livros, artigos e empresas, que possuem esta problemática, visando detectar as particularidades necessárias para implementação destes procedimentos SQL.

Posteriormente, foi feito um levantamento sobre os metadados necessários para capturar a semântica utilizada na criação de procedimentos de carga de dados, ou seja, quais os dados que descrevem este tipo de procedimento e podem permitir geração automática para diversas cargas, levando em consideração os diferentes tipos de armazenamento de histórico para os atributos nas dimensões. Estes metadados serão descritos na próxima seção.

Como diferentes empresas trabalham com diferentes tipos de Sistemas Gerenciadores de Banco de Dados, foi viável conceber uma abordagem para dar suporte a diferentes plataformas, considerando vários dialetos da linguagem SQL. Em outras palavras, processos e códigos já desenvolvidos para um dialeto podem ser reutilizados para outros, uma vez que possuem características e funcionalidades comuns.

A partir da obtenção destas informações, a abordagem pôde ser automatizada. Esta automatização se deu através de um *framework* (vide seção 3.2), para satisfazer os diferentes dialetos SQL, e de uma ferramenta, FGCod, que estende o mesmo (vide seção 3.3).

3.1.1 Definição de Metadados

Nesta seção, são apresentados os passos que foram utilizados para a definição dos metadados necessários para a geração automática de código. Cada tipo de atributo de dimensão recebeu o seguinte tratamento: a) Inicialmente foi definido o processo que traduz as atividades ou tarefas necessárias para o povoamento de dados; b) Considerando o Diagrama de Processos (SMITH; FINGAR, 2003) que representa o fluxo de dados, foi criado um pseudocódigo para representar o código que deveria ser gerado pela ferramenta; c) Com base no pseudocódigo e nas suas possíveis variações, foi realizada uma análise para identificar os metadados necessários.

A Figura 4 mostra o Esquema Conceitual dos Metadados consolidado. Nas seções posteriores será exibido este Esquema por partes, explicando os metadados utilizados para cada Tipo de armazenamento de histórico. A modelagem não foi feita usando subtipos para cada tipo diferente de atributo da Dimensão, todavia o modelo atende aos objetivos da aplicação, que tem o controle sobre os dados.

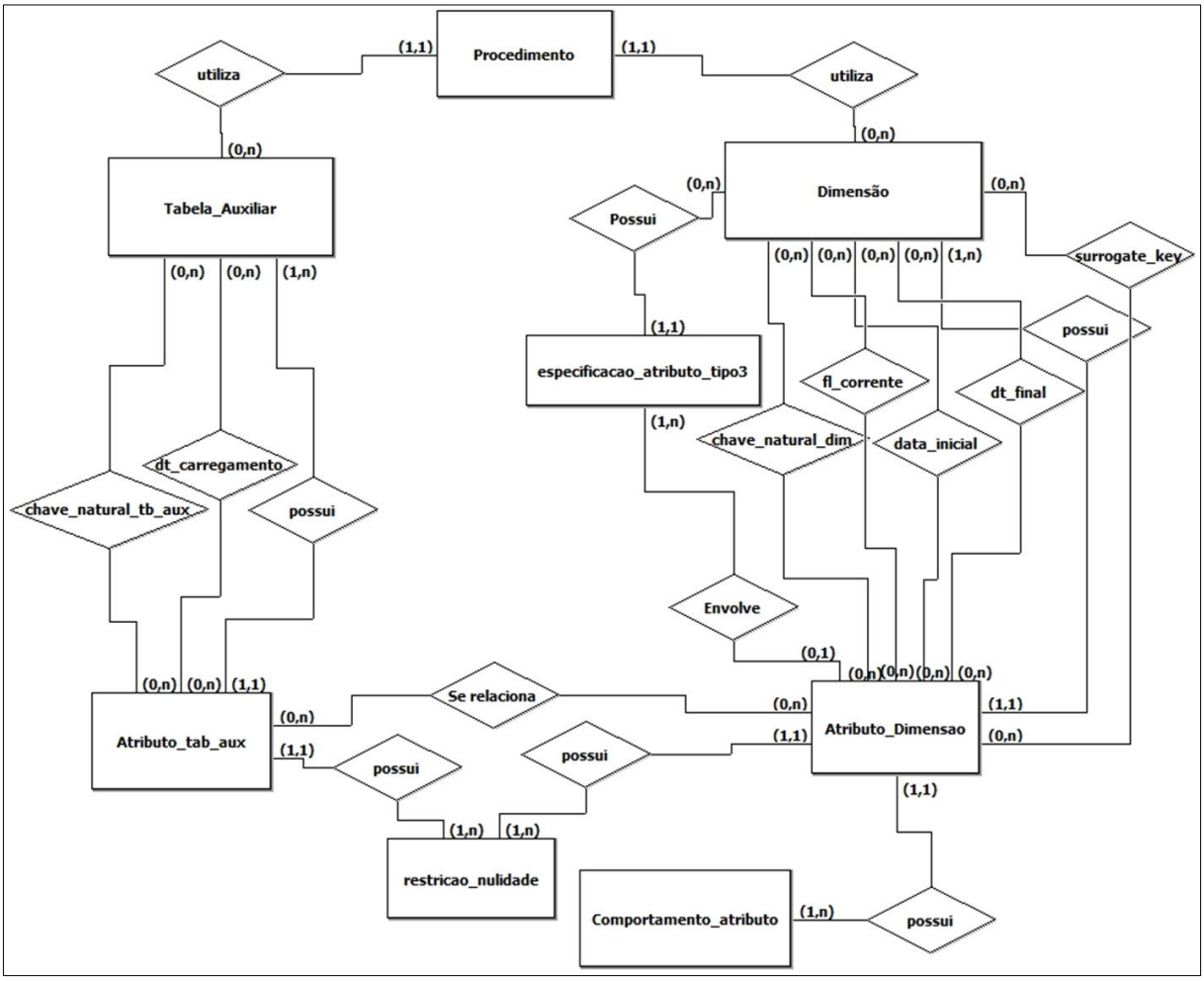

Figura 4: Esquema Conceitual dos Metadados consolidado.

Tipo 1

A Figura 5 representa o Diagrama de Processo do algoritmo para atributos de uma dimensão com comportamento do tipo 1.

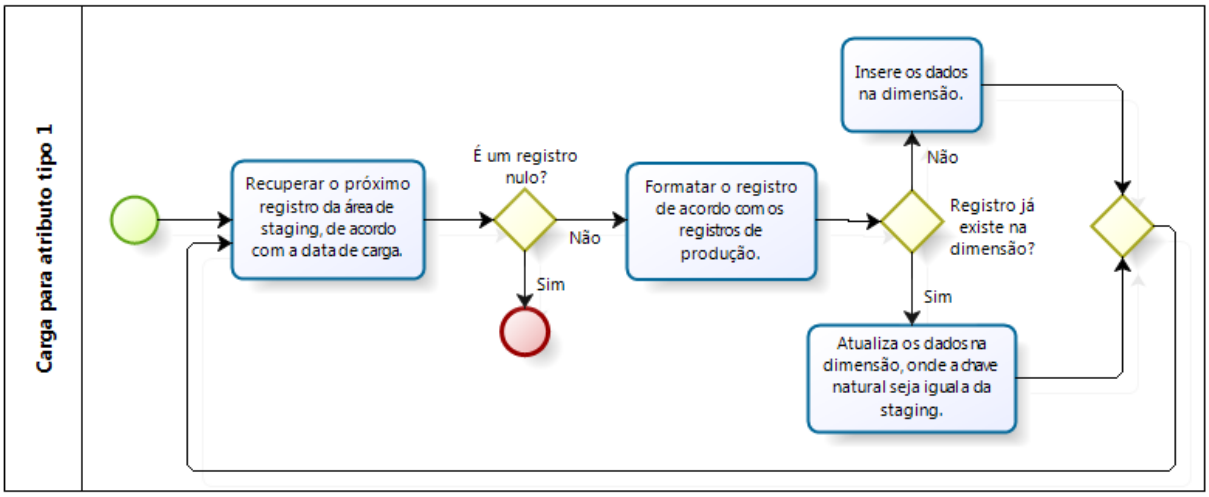

Figura 5**:** Diagrama de Processo do algoritmo de povoamento para atributos do tipo 1.

Inicialmente, o povoamento busca recuperar o registro da Área de *Staging* de acordo com sua data de carregamento. Ao obtê-lo verifica se esse registro é nulo. Caso afirmativo, não há mais atributos na tabela da Área de *Staging* e o carregamento é encerrado. Se negativo, será formatado o registro para ser inserido na dimensão. Nesse momento é feita uma verificação para saber se o registro já está contido na dimensão. Se já estiver, os dados da dimensão são atualizados de acordo com o registro examinado da Área de *Staging*. Caso contrário, insere o registro na dimensão.

A Figura 6 contém um pseudocódigo para carga de atributos do tipo 1. Na primeira linha, é apresentada a assinatura do procedimento. Nessa assinatura, podemos observar a utilização de um parâmetro formal que representa a data utilizada com restrição para recuperação de dados na Área de *Staging*. Em seguida, na linha 3, há a declaração de variáveis que condizem com os atributos da tabela auxiliar. Em 5 há um laço de repetição que será verdadeiro enquanto houver registros na Área de *Staging*, onde a data de carga seja igual a @DATACARGA. Para todo registro examinado pelo laço, o procedimento verifica, na linha 9, se esse já está contido na dimensão. Se verdadeiro, atualiza o registro com os dados mais recentes, de acordo com sua chave natural, leia-se chave primária. Caso contrário, os dados serão inseridos na dimensão.

```
1  CRIAR PROCEDIMENTO Nome_procedimento (@DATACARGA DATETIME)
2  INÍCIO PROCEDIMENTO
3      DECLARAR VARIÁVEIS QUE CORRESPONDEM AOS ATRIBUTOS DA TABELA AUXILIAR
4
5      PARA TODA LINHA DA TABELA AUXILIAR CUJA DATA DE CARGA É IGUAL À DATA PASSADA COMO PARÂMETRO
6      FAÇA
7          INÍCIO
8
9              SE EXISTIR O REGISTRO ATUAL NA DIMENSÃO
10             ENTÃO
11                 ATUALIZA O REGISTRO DA DIMENSAO COM OS DADOS DA TABELA AUXILIAR
12                 DE ACORDO COM SUA CHAVE NATURAL
13             SENÃO
14                 INSERE O REGISTRO, DA TABELA AUXILIAR, NA DIMENSÃO
15
16         FIM
17
18 FIM PROCEDIMENTO
```

Figura 6: pseudocódigo do procedimento para povoamento de atributo tipo 1.

Em todos os procedimentos, a carga será feita de acordo com a chave natural (chave primária), pois, caso haja, no ambiente operacional, alguma alteração na chave primária, esta será feita de forma manual, diretamente no banco, logo, esta mudança no DW também será uma operação manual inerente, ou até mesmo criando outro registro na tabela, mantendo as chaves anteriores e atuais.

Na primeira linha da Figura 6 é definida a criação do procedimento passando como parâmetro a data de carga. O nome dado ao procedimento e o esquema a que pertence são os primeiros metadados identificados. Em 3 é declarada as variáveis que condizem aos atributos da tabela auxiliar. Os metadados que compõem esta tabela e seus atributos são explicados mais adiante, assim como os elementos que formam as dimensões, essenciais para atualização de seus registros, como visto nas linhas 11 e 12.

Em uma tabela auxiliar, os metadados fundamentais são o nome e esquema, assim como os atributos que a constituem. Esses atributos são compostos por: chave natural; nome do atributo; tipo dos dados; tamanho; precisão; escala; restrição de nulidade; atributo que representa data de carregamento. Para a dimensão, os metadados são: *Surrogate Key*, chave natural; nome do atributo; tipo dos dados; tamanho; precisão; escala e restrição de nulidade. Este último, contido em ambas as tabelas serve para indicar se o atributo é ou não um registro nulo.

A dimensão e a tabela auxiliar são cruciais para o funcionamento do procedimento. Dessa forma, este irá conter essas duas tabelas, assim como seus relacionamentos. Afinal, é necessário identificar qual atributo da tabela auxiliar corresponde ao atributo da dimensão, para realizar as atualizações dos registros.

Para contemplar esses metadados, foi criado um esquema conceitual que pode ser visto na Figura 7. Nesta, em (a), estão contidos os metadados para o procedimento. Por fim, em (b) e (c) estão os metadados da tabela auxiliar e dimensão, respectivamente.

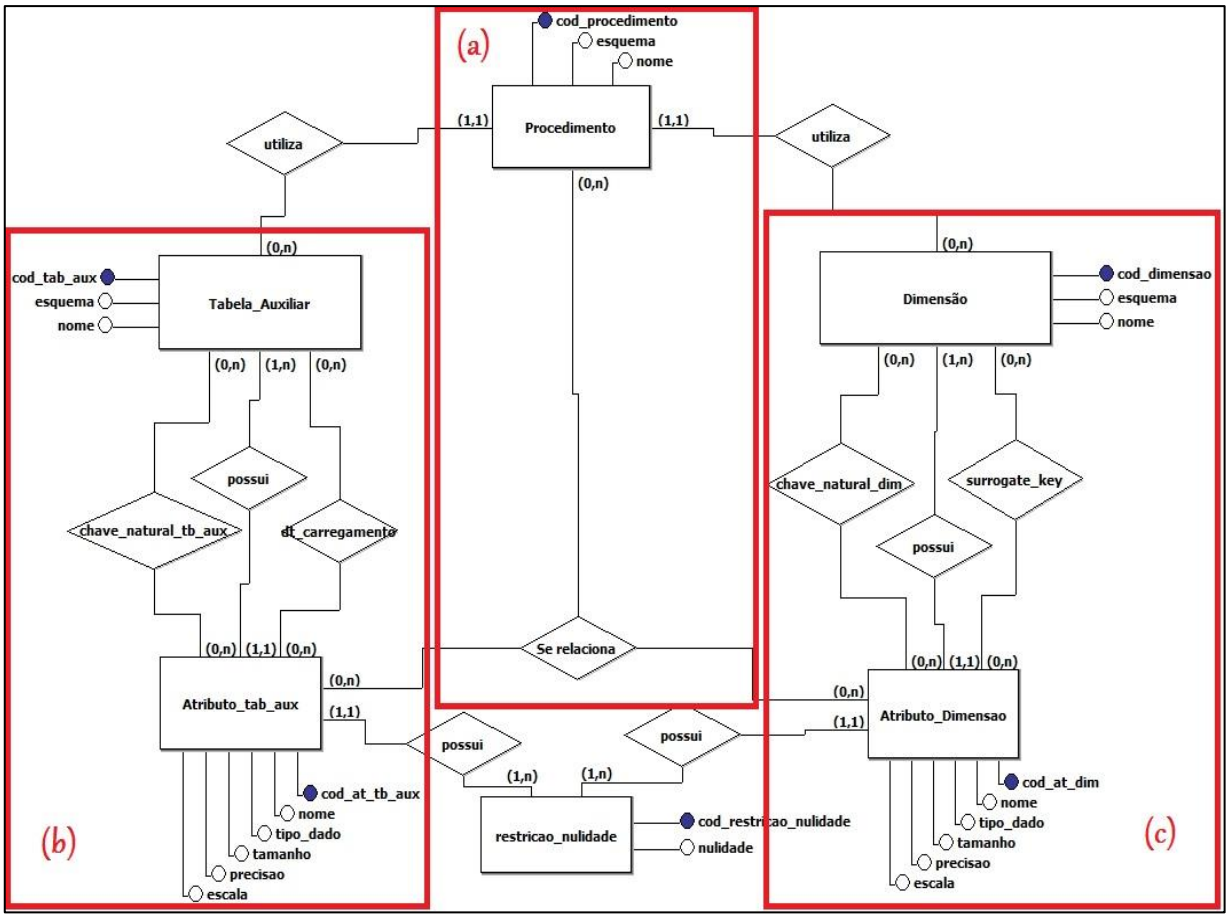

Figura 7: Esquema Conceitual dos metadados – para o procedimento, tabela auxiliar e dimensão – identificados para o povoamento de atributo tipo 1.

Tipo 2

O Diagrama de Processo para o algoritmo de povoamento de atributos do tipo 2 pode ser visto na Figura 8.

Assim como no Diagrama de Processo anterior, serão recuperados e preparados os registros provenientes da Área de *Staging*. Se o registro não estiver contido na dimensão, então é gerada uma nova linha para a mesma, contendo os dados provenientes da tabela auxiliar, e retornará para o começo do processo. Caso contrário, compara os dados correntes da dimensão com o da área auxiliar. Se houve alteração, o procedimento gera um novo registro corrente para a dimensão e o antigo recebe o valor de não corrente. Caso o registro tenha se mantido o mesmo, então irá para o início do processo.

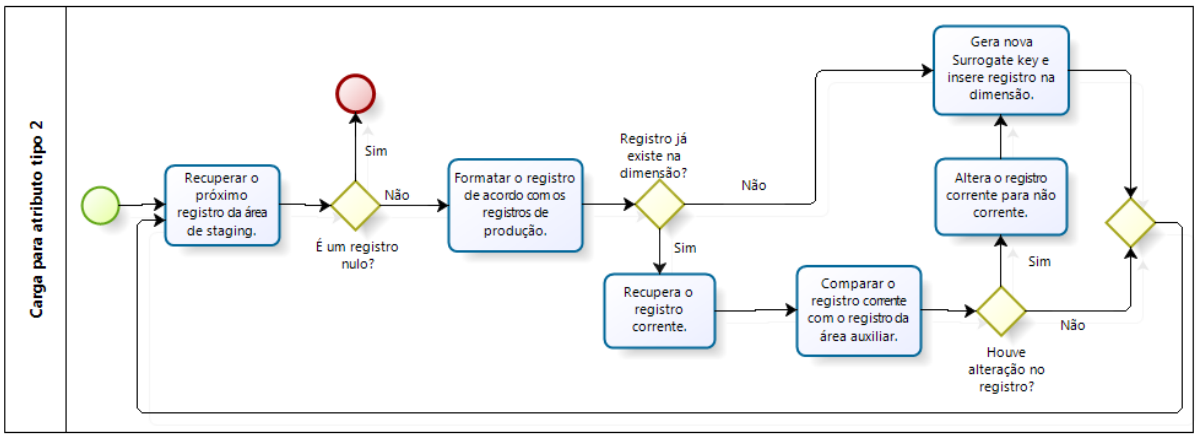

Figura 8: Diagrama de Processo do algoritmo de povoamento para atributos do tipo 2.

A Figura 9 apresenta um pseudocódigo de um procedimento para povoamento de atributos do tipo 2. A diferença desse código para o da Figura 6 são as linhas de 9 a 20. Em 9 verifica se o registro, proveniente da tabela auxiliar, já se encontra na dimensão. Caso não tenha o registro, então insere o novo na dimensão (linha 20). Caso esteja contido, verifica se registro corrente da dimensão é diferente do novo registro, que é proveniente da tabela auxiliar. Se for diferente, altera o registro corrente da dimensão para não corrente e insere o novo como sendo corrente, visto nas linhas 15 e 16.

```
1  CRIAR PROCEDIMENTO Nome_procedimento (@DATACARGA DATETIME)
2  INÍCIO PROCEDIMENTO
3      DECLARAR VARIÁVEIS QUE CORRESPONDEM AOS ATRIBUTOS DA TABELA AUXILIAR
4
5      PARA TODA LINHA DA TABELA AUXILIAR CUJA DATA DE CARGA É IGUAL À DATA PASSADA COMO PARÂMETRO
6      FAÇA
7          INÍCIO
8
9              SE O REGISTRO DA TABELA AUXILIAR JÁ ESTIVER NA DIMENSÃO
10             ENTÃO
11                 INÍCIO ENTÃO
12                     SE O REGISTRO CORRENTE DA DIMENSÃO FOR DIFERENTE DO REGISTRO
13                     PROVENIENTE DA TABELA AUXILIAR DE ACORDO COM SUA CHAVE NATURAL.
14                     ENTÃO
15                         ALTERA O REGISTRO CORRENTE DA DIMENSÃO PARA NÃO CORRENTE
16                         E INSERE O NOVO REGISTRO, DA TABELA AUXILIAR, NA DIMENSÃO.
17
18                 FIM ENTÃO
19             SENÃO
20                 INSERE O REGISTRO, DA TABELA AUXILIAR, NA DIMENSÃO.
21
22         FIM
23
24 FIM PROCEDIMENTO
```

Figura 9: Pseudocódigo do procedimento para povoamento de atributos do tipo 2.

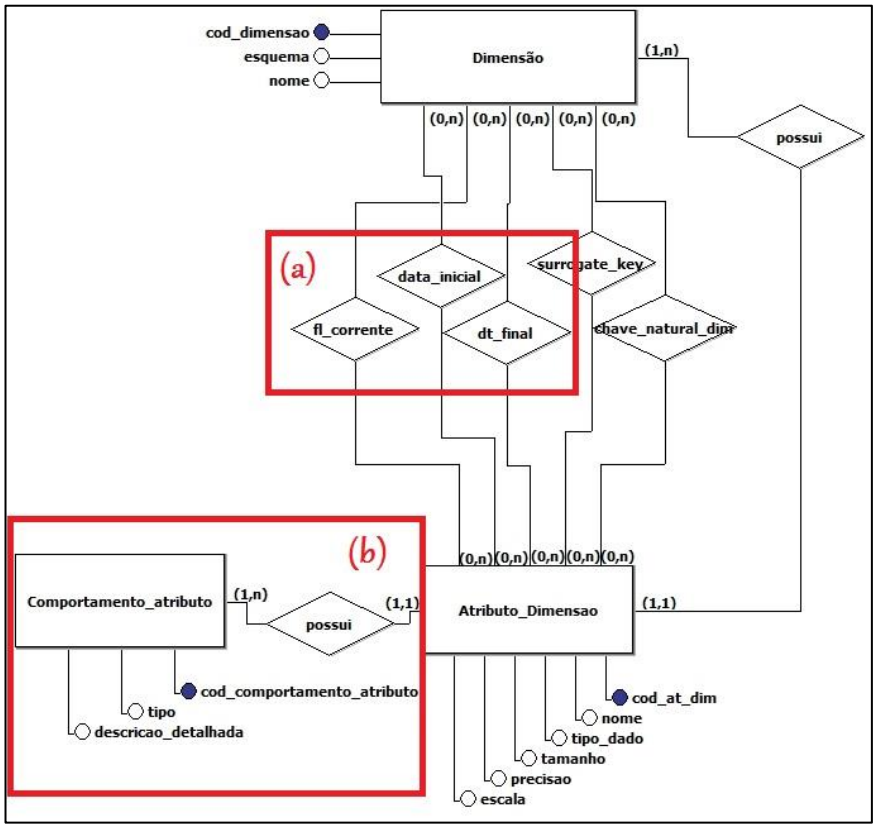

Figura 10: Esquema Conceitual dos metadados identificados, que contemplam a dimensão, para o povoamento de atributo tipo 2.

Os metadados da dimensão que contemplam o povoamento de uma dimensão com atributos do tipo 2 são – além dos identificados no tipo 1 – os seguintes: data inicial, data final e flag corrente (vistos em (a) da Figura 10); comportamento do atributo. Este último surgiu a partir do momento que houve a necessidade de coletar os metadados para os diversos tipos de atributos de povoamento em uma dimensão. Logo, tornou-se necessário criar uma entidade que armazenasse o comportamento do atributo, que corresponde ao tipo 0, 1, 2 ou 3. Essa entidade pode ser vista em (b) da Figura 10.

Tipo 3

Na Figura 11, está contido o Diagrama de Processo do algoritmo para povoamento de atributos do tipo 3 da dimensão.

Inicialmente, o processo para povoamento busca recuperar e formatar os registros não nulos provenientes da Área de *Staging*. Posteriormente, verifica se o registro já se encontra na dimensão. Se não existir, insere o novo registro na dimensão e recupera o próximo registro da área auxiliar. Caso contenha, recupera o registro atual pela chave natural da dimensão e compara com o da *Staging*. Se não tiver alteração, irá retornar para o início do processo. Caso contrário, ocorrerá a atualização dos atributos que correspondem ao tipo 3. O atributo

antepenúltimo atribui o valor do penúltimo que receberá o do atual, e por fim, este último altera seu valor para o mais recente vindo da tabela auxiliar.

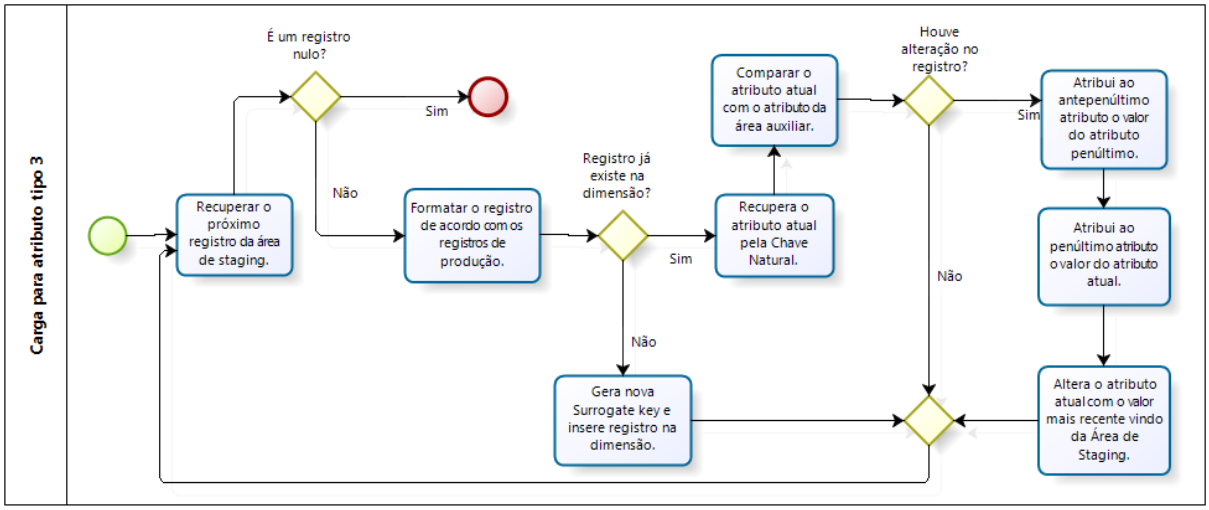

Figura 11: Diagrama de processo do algoritmo para povoamento de atributos do tipo 3.

O código gerado para este tipo de povoamento pode ser visto na FIGURA 12. Após a criação do procedimento e da declaração de variáveis, o código verifica para toda linha da tabela auxiliar cuja data de carga é igual à data passada por parâmetro. Caso seja verdadeiro, confere se o registro da tabela auxiliar já está contido na dimensão (linha 9). Se já estiver, atribui os valores que possuem comportamento tipo 3 às variáveis do procedimento. Após isso, na linha 15, compara o registro atual do atributo tipo 3 com o valor atual da tabela auxiliar. Se for diferente, atualiza os registros que correspondem ao tipo 3 de acordo com os novos registros.

```
1  CRIAR PROCEDIMENTO Nome_procedimento (@DATACARGA DATETIME)
2  INÍCIO PROCEDIMENTO
3      DECLARAR VARIÁVEIS QUE CORRESPONDEM AOS ATRIBUTOS DA TABELA AUXILIAR
4
5      PARA TODA LINHA DA TABELA AUXILIAR CUJA DATA DE CARGA É IGUAL À DATA PASSADA COMO PARÂMETRO
6      FAÇA
7          INÍCIO
8
9              SE O REGISTRO DA TABELA AUXILIAR JÁ ESTIVER NA DIMENSÃO
10             ENTÃO
11                 INÍCIO ENTÃO
12                     ATRIBUIR ÀS VARIÁVEIS OS VALORES QUE POSSUEM COMPORTAMENTO TIPO 3,
13                     A PARTIR DA CHAVE NATURAL DA TABELA AUXILIAR.
14
15                     SE O REGISTRO ATUAL DO ATRIBUTO TIPO 3 DA DIMENSÃO FOR DIFERENTE DO REGISTRO
16                     PROVENIENTE DA TABELA AUXILIAR DE ACORDO COM SUA CHAVE NATURAL.
17                     ENTÃO
18                         ATUALIZA OS REGISTROS QUE CORRESPONDEM AO TIPO 3
19                         DE ACORDO COM A TABELA AUXILIAR.
20
21                 FIM ENTÃO
22             SENÃO
23                 INSERE O REGISTRO, DA TABELA AUXILIAR, NA DIMENSÃO.
24
25         FIM
26
27 FIM PROCEDIMENTO
```

Figura 12: Pseudocódigo do procedimento para povoamento de atributo do tipo 3.

Com base no código, foi identificado que para conceber os metadados para o atributo tipo 3, seria necessário adicionar uma entidade que servisse para armazenar os atributos atual, penúltimo e antepenúltimo de uma dimensão. O nome dessa entidade (visto em vermelho na Figura 13) é a especificação do atributo tipo 3.

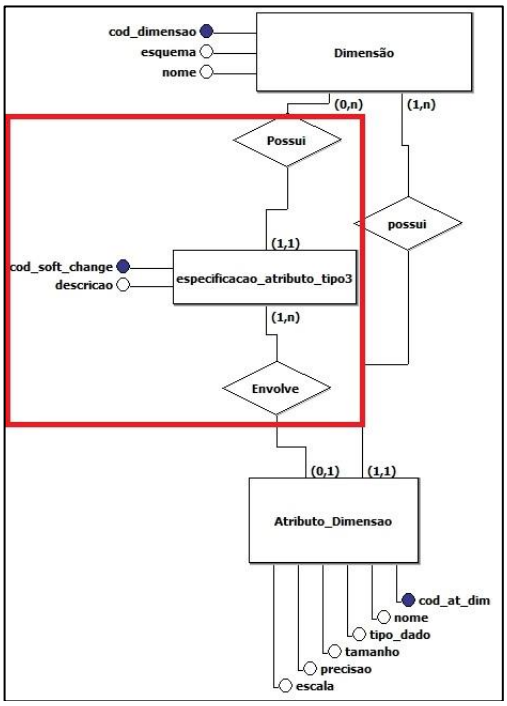

Figura 13: Esquema Conceitual dos metadados identificados, que contemplam a dimensão, para o povoamento de atributo do tipo 3.

Tipo 0

No início do processo para carga de atributos do tipo 0 (Figura 14), o procedimento busca um registro na tabela da *Staging*, de acordo com sua data de carregamento. Após isso, verifica se esse é um registro com valores nulos. Caso não seja, o procedimento preparara o dado e irá conferir se o registro já existe na dimensão. Se não possuir o valor, este irá ser persistido na dimensão. Caso já possua, o procedimento ignora a atualização e buscará recuperar o próximo registro da tabela auxiliar.

Diante do que pode ser analisado no processo desse tipo do atributo, foi criado o pseudocódigo da Figura 15. Nesse, a cada iteração, que ocorre na linha 5, o código verifica se o registro já foi inserido na dimensão. Caso não esteja, o novo registro vindo da área auxiliar será inserido na dimensão (linha 11). Caso contrário, esse registro será ignorado. Visto que esse trecho do código já se encontra nos demais pseudocódigos apresentados anteriormente, não houve descoberta de novos metadados para o projeto.

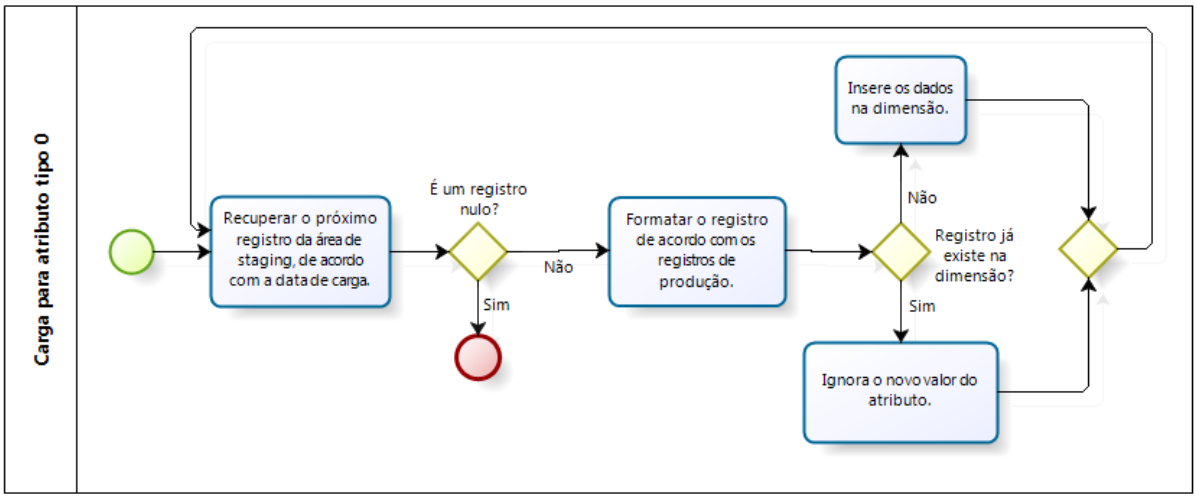

Figura 14: Diagrama de Processo do algoritmo para povoamento de atributo do tipo 0.

```
1  CRIAR PROCEDIMENTO Nome_procedimento (@DATACARGA DATETIME)
2  INÍCIO PROCEDIMENTO
3      DECLARAR VARIÁVEIS QUE CORRESPONDEM AOS ATRIBUTOS DA TABELA AUXILIAR
4
5      PARA TODA LINHA DA TABELA AUXILIAR CUJA DATA DE CARGA É IGUAL À DATA PASSADA COMO PARÂMETRO
6      FAÇA
7          INÍCIO
8
9              SE O REGISTRO DA TABELA AUXILIAR NÃO ESTIVER NA DIMENSÃO
10             ENTÃO
11                 INSERE O REGISTRO, DA TABELA AUXILIAR, NA DIMENSÃO.
12
13         FIM
14
15 FIM PROCEDIMENTO
```

Figura 15: Pseudocódigo do procedimento para povoamento de atributos do tipo 0.

Nos experimentos serão utilizados apenas os tipos 1 e 2, por serem os tipos mais utilizados nas empresas. Não foram encontrados *surveys* sobre quais os tipos mais utilizados, contudo, empresa parceira relatou que é a prática em todas as empresas atendidas, cerca de 10. Além disso, o tipo 2 pode fornecer as mesmas consultas fornecidas pelo tipo 3, desde que seja eliminado o histórico não mais utilizado. E o tipo 0 não se torna muito interessante, já que as empresas necessitam dos dados atualizados.

3.2 Implementação do Framework para Geração Automática de Código

Esta seção tem como objetivo descrever como foi feito o desenvolvimento do *framework* de suporte a ferramentas de geração de código automática para ambientes de apoio à decisão. Este *framework* implementará a funcionalidade de geração de procedimentos SQL, que terão como função a carga de dados da área de *Staging* para o DW.

3.2.1 Análise de Domínio

O escopo de domínio do *framework GerarProcedimentoSQL* é abranger aplicações que necessitam gerar procedimentos SQL para cargas de dados, sendo estas da área de *Staging* para dimensões de um DW, de tal forma que utilizem da reutilização de uma estrutura já existente, ou seja, estendendo o *framework*.

O modelo de classes desta fase está representado abaixo, na Figura 16.

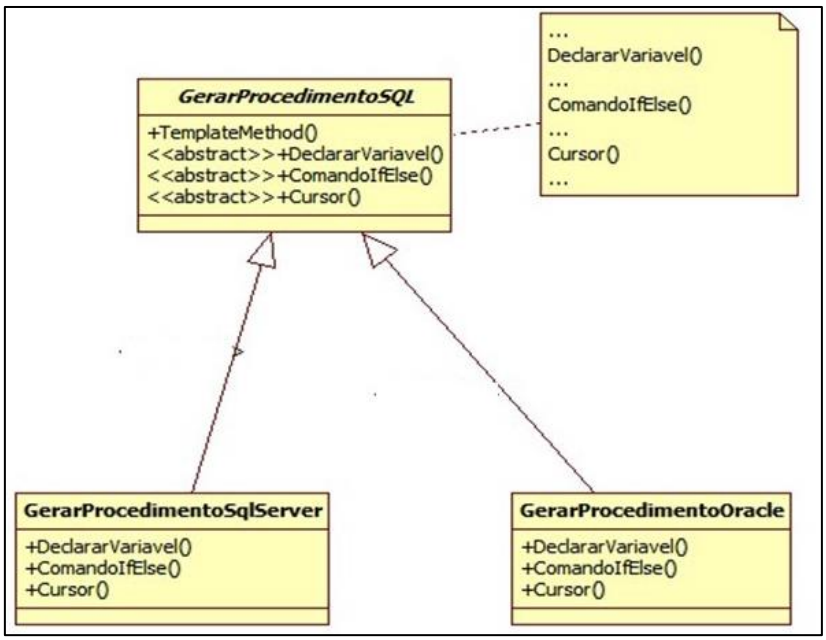

Figura 16: Modelo de Classes do *framework*.

3.2.2 Levantamento de Requisitos e Análise

Prosseguindo com o processo de desenvolvimento do *Framework*, passamos agora para fase de Levantamento de Requisitos e Análise. Nesta fase serão descritos o modelo de Requisito e o Modelo de Análise.

Modelo de Requisitos

Para o Modelo de Requisitos será criada a Especificação dos Requisitos.

O *Framework* deve atender às seguintes especificações de requisito:

- Requisitos Funcionais

  R1: Suportar criação de procedimentos para dimensões com atributos do tipo 1;

  R2: Suportar criação de procedimentos para dimensões com atributos do tipo 2;

  R3: Suportar criação de procedimentos para dimensões com atributos do tipo 3;

  R4: Encapsular lógica independente do banco;

- Não funcionais

  R5: O *framework* deve manter a consistência dos metadados.

  R6: A integridade dos metadados deve ser mantida.

Modelo de Análise

O modelo de análise sofreu algumas alterações, em relação ao que foi criado na fase de análise de domínio. Essas alterações foram frutos do detalhamento do modelo, ou seja, os objetos e suas relações foram modelados de forma minuciosa.

O modelo de classes desta fase está representado na Figura 17.

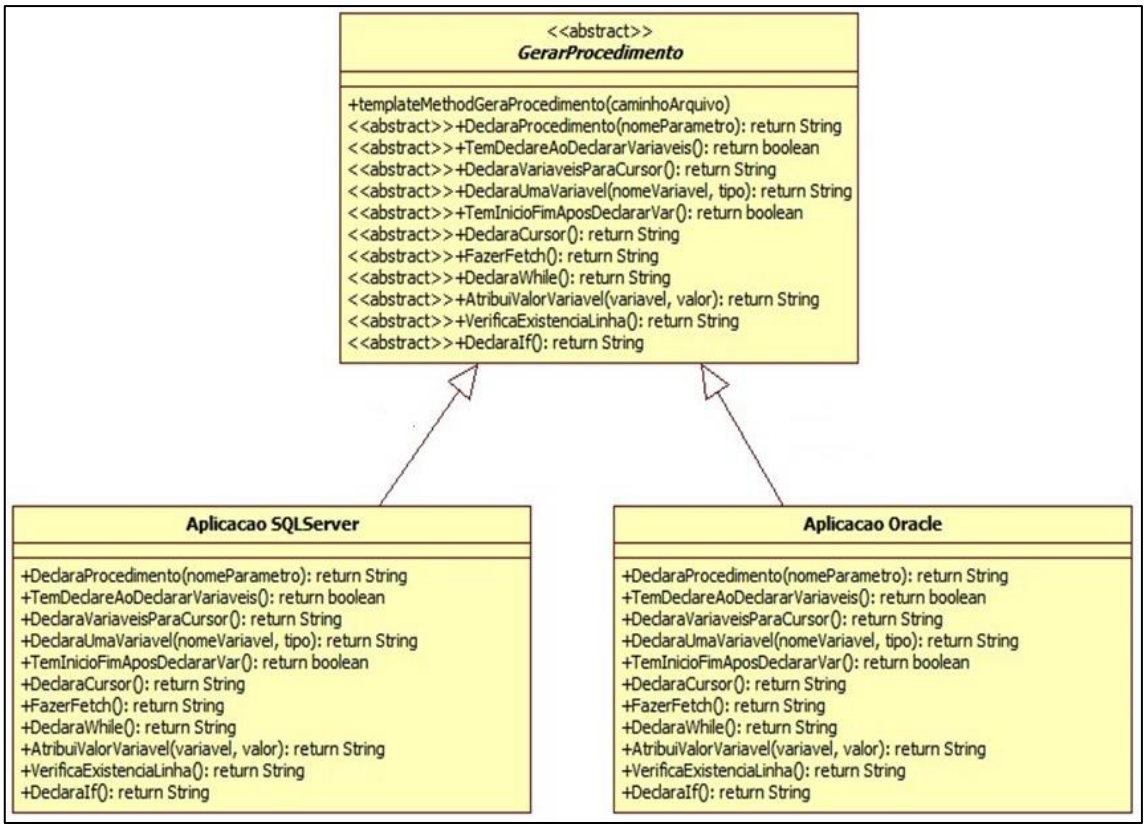

Figura 17: Modelo de Classes, com melhor modelação do *framework.*

3.2.3 Projeto

O projeto do *Framework GerarProcedimentoSQL* é formado pelo projeto arquitetural, no qual será demonstrado a forma que se dispõe a arquitetura do *framework*, e pelo projeto de baixo nível, o qual é responsável por detalhar o modelo conceitual, mostrando as classes que serão implementadas e a interação entre os objetos.

3.2.3.1 Projeto Arquitetural

O projeto Arquitetural foi desenvolvido com o intuito de documentar as decisões de alto nível. Sendo identificadas as camadas, módulos e suas interdependências. Visando atender aos requisitos não-funcionais, detectados na fase de análise. Além disso, mostra a tecnologia utilizada para comunicação entre camadas.

A arquitetura do *Framework GerarProcedimentoSQL* está representada em três camadas: Interface, Aplicação e Dados.

A camada que interage com o usuário final é a camada de Interface. É nela que o usuário poderá escolher para qual banco de dados, diante dos já implementados, ele vai gerar o procedimento. Além disso, o usuário escolherá qual o destino em que o arquivo, com a extensão .sql, será armazenado. É este arquivo que conterá o procedimento em SQL, o qual será utilizado para realizar a carga de uma tabela auxiliar, da Área de *Staging* para uma dimensão no DW. Nesta etapa, os metadados necessários para criação do procedimento já devem estar armazenados no Banco de Dados, pois neste momento, serão informadas quais tabelas auxiliar e dimensão farão parte do procedimento.

A camada de aplicação contém toda regra de negócio necessária para funcionamento do *framework*. Este foi dividido em duas partes, uma classe abstrata, na qual é feita toda a lógica e sequencia de métodos em comum a serem chamados, para formação do procedimento, e uma classe concreta, na qual, tem o objetivo de realizar as instanciações para captura dos metadados utilizado nas classes que fazem extensão da classe abstrata.

A camada de dados é responsável por fornecer os metadados necessários para montagem do procedimento.

Além dos módulos existentes, o projeto arquitetural documenta os pontos de extensão existentes no *Framework GerarProcedimentoSQL*. Desta forma facilita o desenvolvimento de novas aplicações que irão estender este *framework*. Estas devem se preocupar em estender a classe abstrata, *AbstractGerarProcedimento*, e também utilizar alguns métodos fornecidos pela classe concreta, *MetodosComplementares*.

3.2.3.2 Menor Nível de Abstração

Este projeto, de menor nível de abstração, define os modelos que auxiliarão na implementação do *framework*. Será ilustrado o diagrama de classes detalhado.

Diagrama de Classes

O diagrama de classes modela as classes, com métodos e objetos a serem utilizados.

A Figura 18 ilustra o diagrama de Classes do Framework em questão.

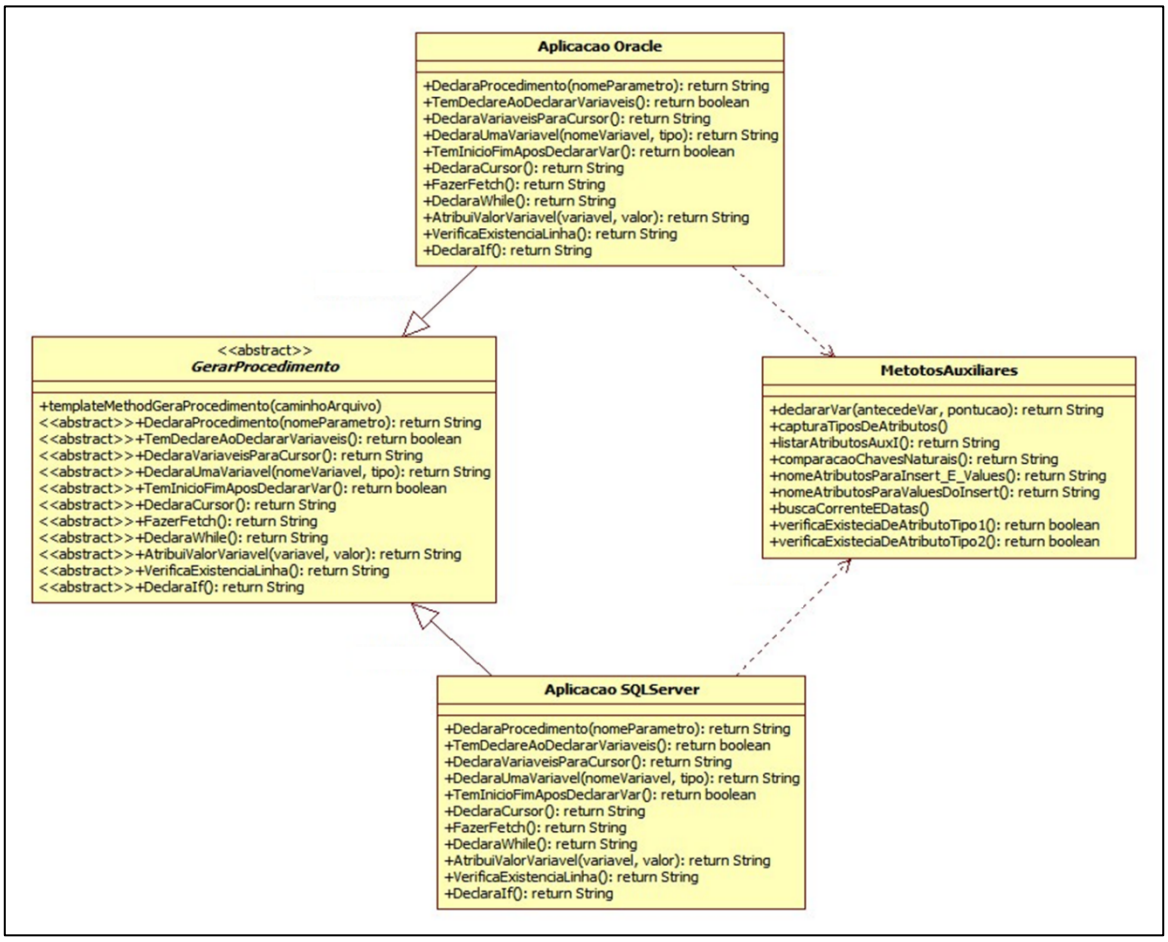

Figura 18: Diagrama de Classes do Framework GerarProcedimentoSQL

Este modelo sofreu alterações, com relação ao definido no Modelo de Análise (ver Figura 17). Inicialmente, foi definido que existiria apenas a classe abstrata, que conteria o método *templateMethodGeraProcedimento*, e as classes concretas das aplicações, que estenderiam a classe abstrata. No entanto, para criação do arquivo.sql, que conterá o procedimento, é preciso carregar os metadados que estão armazenados no Banco de Dados. Como a classe abstrata não pode instanciar objetos, elas estavam sendo feitas separadamente, ou seja, cada classe concreta faria as implementações com as instancias necessárias. Então, as diferentes classes estavam implementando métodos que continham a mesma funcionalidade, e a classe abstrata se responsabilizava praticamente apenas em ordenar a chamada destes métodos.

Desta forma, não se via muita viabilidade na criação do *framework*, já que existiriam funcionalidades idênticas, sendo implementadas mais de uma vez. A duplicação de código é desnecessária e fere o objetivo inicial, o qual diz que as classes concretas que estenderão o *framework* implementem apenas as características que as diferenciam.

Para tentar resolver este problema, sem que houvesse a necessidade de muita mudança, foi criada uma nova classe concreta, *MetodosComplementares*, para auxiliar a classe abstrata, implementando assim os métodos que contêm as funcionalidades idênticas, podendo ser chamada pelas classes que estenderão a classe *AbstractGerarProcedimento*.

3.2.4 Implementação

Na fase implementação do *framework*, será materializado, em código, tudo o que foi modelado nas fases anteriores, principalmente nos modelos do projeto, os quais estão descritos de forma mais aprofundada.

As classes implementadas são as seguintes:

- *AbstractGerarProcedimento*: classe abstrata que deve ser estendida para a reutilização das funcionalidades desenvolvidas;
- *MetodosComplementares*: classe concreta que contem instanciações para obtenção dos metadados necessários à geração do procedimento em SQL.

Para reutilização do *Framework GerarProcedimentoSQL*, foram implementadas duas classes, uma contendo as particularidades do dialeto da linguagem SQL da Oracle (PL/SQL) e outra contendo as particularidades do SQLServer (Transact-SQL, ou apenas T-SQL).

Estas classes devem estender a classe abstrata *AbstractGerarProcedimento*, ela conterá o método *templateMethodGeraProcedimento*, o qual tem como parâmetro o caminho em que o arquivo.sql deve ser salvo. Este e os outros parâmetros (nome do procedimento e dialeto da linguagem SQL) serão informados em uma ferramenta já existente.

3.2.5 Testes

Após da realização da implementação, deve-se garantir que o *framework* está se comportando como realmente deve. Para isso, foram realizados testes a partir das classes concretas, implementando as funcionalidades específicas, à medida que o *framework* era construído, com a finalidade de constatar se o comportamento do mesmo correspondia ao resultado esperado.

3.2.6 Estudo de Caso

Nesta seção, são abordadas as atividades necessárias para a execução do estudo de caso realizado. Na Seção 3.2.6.1, é apresentado o objetivo do estudo realizado, em seguida, na seção 3.2.6.2, é apresentada a definição do planejamento com a seleção dos bancos de dados utilizados e os casos de uso criados. Por fim, na Seção 3.2.6.3, é apresentado o processo de operação que consiste na execução e validação dos dados processados.

3.2.6.1 Definição de Objetivo

A realização do estudo de caso tem como objetivo principal a validação dos resultados emitidos pelo *framework* produzido neste trabalho. Contudo, para atingir este objetivo, é necessária a realização dos seguintes objetivos específicos:

- Escolher pelo menos dois dialetos da linguagem SQL;
- Implementar classes que atendam às especificidades de cada dialeto, utilizando o *framework*;
- Fazer execução do *framework*, a fim de que o mesmo gere os códigos para os dialetos implementados.
- Criar ambiente, nos bancos de dados escolhidos, para testar o código gerado;
- Verificar e validar os resultados obtidos;
- Realizar reajustes no *framework*, se necessário.

Definindo os objetivos, o planejamento está descrito a seguir.

3.2.6.2 Planejamento

Para alcançar os objetivos listados anteriormente é preciso traçar uma estratégia de execução. Começando pela seleção dos bancos de dados, ou seja, os dialetos da linguagem SQL seguindo pela implementação das classes utilizando o *framework* e testando se o código gerado faz a carga desejada.

Os dialetos que serão utilizados para teste do *framework* já foram definidos desde o início do projeto, será o dialeto do SQL Server, T-SQL, e o do Oracle, PL/SQL. Estes foram escolhidos devido à grande popularidade e pelo fato de ambos terem características diferentes

entre si. Muitos outros dialetos SQL existentes assemelham-se aos mesmos, ajudando, desta forma, na veracidade dos testes, a serem realizados.

Para implementação das classes, que utilizarão o *framework*, devem ser observados os pontos de extensão e quais métodos específicos a classe, de cada dialeto, precisa implementar.

Após implementar as classes e verificar a possibilidade da geração de código, é preciso testá-los. Para isto, serão criados cenários de uso para cada banco de dados. Ou seja, serão criadas tabelas auxiliares e suas respectivas dimensões para realizar a execução do procedimento com as mesmas. Inicialmente serão criados ambientes que tenham armazenamento de histórico até o tipo 2.

Diante disto, será possível verificar se foi possível plugar uma classe para implementação de um determinado dialeto SQL e, caso resultado positivo, se, com a utilização do procedimento gerado, a carga dos dados foi realizada de forma correta, validando ou não a eficiência do *framework*. Com o resultado, caso necessário, o *framework* poderá sofrer ajustes.

3.2.6.3 Operação

A operação para a realização do estudo de caso foi dividida em duas etapas. A etapa de Execução e a etapa de Validação dos dados. Cada uma delas é descrita a seguir.

EXECUÇÃO

Para realizar a execução dos objetivos especificados foi necessário, primeiramente, implementar as duas classes concretas, com especificidades do T-SQL e PL/SQL. Estas estenderiam a classe abstrata, na qual, já estava implementado os passos comuns aos dois dialetos. Para isto, foi preciso analisar o ponto de extensão do *framework* e seus métodos abstratos, os quais necessitariam de implementação nas duas classes concretas, para verificar se realmente seria possível a implementação para tal dialeto utilizando o *framework.*

A implementação seguiu de acordo com o exemplo demonstrado na Figura 19, Figura 20 e Figura 21:

```
public abstract class AbstractGerarProcedimento {

public abstract String DeclaraCursor();
...

public final void templateMethodGeraProcedimento(String caminhoArquivo){

...

String declaracaoCursor = DeclaraCursor();
...

}
}
```

Figura 19: Classe Abstrata – *AbstractGerarProcedimento*.

```
public class GerarOracle extends AbstractGerarProcedimento{

private Procedimento procedimento;
...

public GerarOracle(Procedimento procedimento, ...){
      this.procedimento   = procedimento;

      ...
}
...
    public String DeclaraCursor(){
        String nomeCursor = "Cursor_"+procedimento.getNome_procedimento();
        return "DECLARE CURSOR "+ nomeCursor+" IS ";
    }
...
}
```

Figura 20: Classe concreta que contem especificações do Oracle.

```
       public class GerarSqlServer extends AbstractGerarProcedimento{

                    private Procedimento procedimento;
                                ...

            public GerarOracle(Procedimento procedimento, ...){
                        this.procedimento   = procedimento;

                                   ...
                                }
                                ...
                    public String DeclaraCursor(){
     String nomeCursor = "Cursor_"+procedimento.getNome_procedimento();
               return "DECLARE "+ nomeCursor+" CURSOR FOR ";
                                }
                                ...
                                }
```

Figura 21: Classe concreta que contem especificações do SQLServer.

Existe ainda a Classe Concreta *MetodosComplementares*, que serve como auxilio para a classe Abstrata *AbstractGerarProcedimento*, podendo ser chamada pela *GerarOracle* ou *GerarSqlServer*. Ela foi chamada, por exemplo, para devolver uma lista de *Strings* contendo os atributos da tabela auxiliar, já com seus respectivos tipos. Esses dados podem ser necessários também para outra classe que queira estender o *framework*. Por este motivo, foi implementado na Classe dos métodos complementares, a fim de evitar a duplicação de código.

VALIDAÇÃO DOS DADOS

Além da validação inicial, realizada na Seção Testes, foi realizada uma validação mais específica durante a execução do estudo de caso. Para esta, foram criados cenários de uso, ou seja, foram criadas tabelas auxiliares e dimensões, em SQL Server e Oracle. Os dados que compõem estas tabelas foram cadastrados na ferramenta *FGCod*. Após cadastro destes metadados, o procedimento pode ser gerado através da mesma.

Abaixo, na Figura 22, é ilustrada a tela da ferramenta responsável por informar os dados que farão parte da estrutura do procedimento a ser gerado.

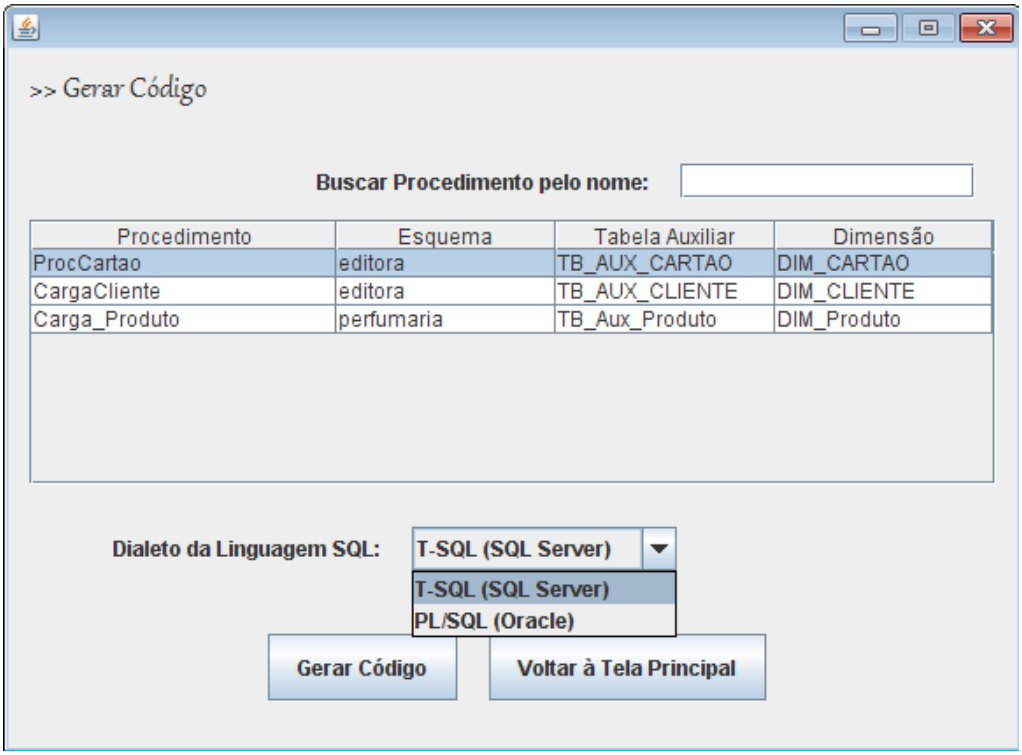

Figura 22: Tela inicial da ferramenta para dar suporte ao *framework*.

Após a geração dos procedimentos, em ambos os dialetos, os mesmos foram executados em seus SGBD's (Sistema Gerenciador de Banco de Dados) a fim de testar se o código gerado atendeu à sua função. Ambos os testes obtiveram êxito, ou seja, a carga dos dados aconteceu de forma correta.

A principal contribuição do *framework* é dar suporte a ferramentas de geração de código automática para geração de procedimentos SQL, os quais fazem carga da área de *Staging* para o DW. Para consolidação e avaliação do processo de extensão do *framework* desenvolvido, foi realizado este estudo de caso, no qual foram escolhidos dois dialetos da linguagem SQL, o T-SQL e o PL/SQL. O desenvolvimento se deu com base no padrão de projeto *Template Method*. Foi construída uma classe abstrata, na qual se encontrava todos os pontos em comum entre tais dialetos da linguagem SQL, para que mais tarde pudesse ser estendida pelas classes que implementariam as especificidades de cada dialeto. Para geração do procedimento, torna-se necessária a obtenção dos metadados, os quais se encontram num banco de dados, que, neste caso, foi preenchido através da FGCod.

O código da FGCod também foi reaproveitado através da criação de uma classe concreta como auxílio à classe abstrata do *framework*. Esta classe instancia e repassa os dados para as classes estendidas.

Considerando as validações de dados descritas nesta seção, ficou evidente a possibilidade do uso do *framework* para dar suporte à geração de código utilizado na realização de cargas de um DW, em diferentes dialetos da linguagem SQL.

3.3 FGCod

A FGCod é uma ferramenta de RAD para aumentar a efetividade do desenvolvimento e manutenção de programas ETL. A mesma foi desenvolvida na linguagem Java, de forma iterativa e incremental. Ela está disponível para *download* em <http://fgcod.wordpress.com/>. Neste *link*, também se encontra o tutorial e pacote experimental para replicação. Diante do *framework* apresentado na seção anterior, extensões da ferramenta podem gerar códigos para outros SGBDs.

A elaboração da ferramenta foi realizada em duas principais etapas. A primeira delas teve como objetivo definir os metadados necessários para gerar o código sql, os quais já

foram descritos na seção Definição de Metadados. Na segunda etapa, foi realizada a construção da ferramenta, que foi desenvolvida de modo iterativo e incremental.

3.3.1 Análise e Projeto da Ferramenta

Após a definição dos metadados, iniciou-se o projeto da ferramenta, no qual foram analisados e definidos quais seriam seus requisitos funcionais e não funcionais. Estes podem ser vistos no Quadro 3.

A ferramenta deve conter, como requisitos funcionais, o gerenciamento dos metadados fundamentais para as tabelas auxiliares e dimensões (com atributos tipos 1, 2 e 3). Torna-se imprescindível, para efetivar a geração de código sql, realizar os relacionamentos dos atributos da tabela auxiliar com os da dimensão. Além de gerenciar os procedimentos de povoamento dimensões.

Quadro 3: Requisitos funcionais e não funcionais da ferramenta.

Requisitos Funcionais	**R1:** Gerenciar os metadados necessários para a Tabela Auxilar. **R2:** Gerenciar os metadados necessários dos seguintes elementos do ambiente de *Data Warehouse:* Dimensões; **R3:** Gerenciar o relacionamento dos atributos da Tabela Auxiliar com os atributos das tabelas correspondentes no ambiente de *Data Warehouse*. **R4:** Gerenciar procedimentos de povoamento de dados para dimensões. **R5:** Gerar o código SQL para os procedimentos.
Requisitos Não Funcionais	**R6:** A ferramenta deve manter a consistência dos metadados. **R7:** A integridade dos metadados, provenientes da base de dados, deve ser mantida. **R8:** O tempo de geração do procedimento não deve ultrapassar 10 segundos.

A concepção da aplicação se deu a partir um processo iterativo e incremental. Cada incremento foi responsável pela geração de código envolvendo um tipo de comportamento para o tratamento de histórico de atributos do tipo 1, 2 e 3.

A cada incremento foram realizados testes com casos de uso para validar as funções sendo incorporadas à ferramenta. Os testes garantiram resultados positivos em relação aos objetivos propostos da ferramenta, que são: redução do tempo e dos números de erros de codificação durante a fase de povoamento de dados em um ambiente de *Data Warehouse*.

3.3.2 Funcionalidades da FGCod

Segue, abaixo, descrição das principais funcionalidades da FGCod.

A Figura 23 ilustra a parte principal da tela inicial da FGCod. Em (1), pode ser realizado o gerenciamento das tabelas auxiliares (tabelas intermediárias que recebem dados dos sistemas transacionais), ou seja, são configurados o nome e esquema da tabela, assim como os atributos que a compõe, suas características e comportamentos. Em (2), acontece o gerenciamento das tabelas de dimensão, onde são configurados os metadados que compõem uma dimensão e seus tratamentos para histórico de dados. No gerenciamento do procedimento, localizado em (3), deve ser informada a tabela auxiliar e a dimensão que irá compor o procedimento, além de criar o relacionamento entre os atributos dessas tabelas. Para finalizar, em (4), o usuário gera código SQL para o procedimento definido em (3).

Figura 23: Tela inicial da FGCod

Em (5) são feitas as configurações de conexão com o banco de dados e também a seleção das bases de dados, que possuem as tabelas necessárias para captura dos metadados que auxiliarão na criação do procedimento. A opção (6) é responsável por manter os procedimentos já criados anteriormente, então, nesta opção será possível selecionar um procedimento acrescentar algum dado para o mesmo, excluir ou até mesmo alterar.

Vale ressaltar que, o módulo em questão da FGCod, trata apenas do processo de geração de código na criação e manutenção de procedimentos para carga de dados. Módulos tais como modelagem gráfica do *workflow* (orquestração), modelagem textual (MAZANEC; MACEK, 2012) e *engine* de execução não estão sendo discutidos neste trabalho.

CAPÍTULO 4

4. EXPERIMENTO I

Este experimento se refere à avaliação do módulo da FGCod para criação automatizada de procedimentos SQL em ambientes de BI.

4.1 Definição e Planejamento do Experimento

Nesta e nas duas próximas seções, 4.2 e 4.3, o conteúdo é apresentado como um processo experimental. O mesmo segue as diretrizes de Wohlin em (WOHLIN et al., 2000). Esta seção irá focar na definição do objetivo e no planejamento do experimento.

4.1.1 Definição do Objetivo

O objetivo deste experimento é avaliar, por meio de um experimento controlado, o uso da geração automática de código para criação de rotinas de povoamento em um ambiente de *Data Warehouse*, através de uma ferramenta de Desenvolvimento Rápido de Aplicações (RAD), a FGCod.

O experimento terá como alvo programadores de processos ETL para ambientes de *Business Intelligence,* com alguma experiência no mercado. O objetivo foi formalizado utilizando o modelo GQM proposto por Basili (BASILI; WEISS, 1984): **Analisar** o uso de uma ferramenta de geração de código automático, na criação de procedimentos SQL, para povoar um ambiente DW, **com a finalidade de** avaliar se a utilização de uma ferramenta de geração automática de código pode substituir a codificação manual, **com respeito à** eficiência e eficácia do processo de geração automática de código, **do ponto de vista de** programadores e gestores de suporte à decisão, **no contexto de** programadores de uma empresa de *BI*.

4.1.2 Planejamento

4.1.2.1 Formulação de Hipóteses

As questões de pesquisa para o experimento que precisam ser respondidas são as seguintes: 1ª) A geração automática de código pode reduzir o tempo dos programadores no processo de criação de procedimentos para povoamento de um DW?; 2ª) A geração automática de código pode reduzir ou eliminar erros na codificação, para criação de procedimentos e nos dados povoados em um DW?

Para avaliar estas questões, serão utilizadas duas métricas como variáveis dependentes: 1ª) Média de tempo, para criação da Codificação Manual e para criação da Codificação Automática; 2ª) Média de grau de criticidade de erros de povoamento, calculada considerando: (a) Registros carregados incorretamente para o DW: (1) Erro e falta de tratamento histórico para atributos das dimensões; (2) Registros duplicados; (3) Não atualização dos dados. (b) Utilização de matriz GUT (sigla para Gravidade, Urgência e Tendência) (MARSHALL et al., 2011) para gerenciamento de problemas através dos erros citados no item (a), bem como cálculo do grau de criticidade.

Tendo os objetivos e métricas definidas, serão consideradas as hipóteses:

HIPÓTESE 1

- H_{0tempo}: A codificação de criação automática e manual tem a mesma eficiência.

$(\mu_{tempoCodificaçãoManual} = \mu_{tempoCodificaçãoAutomática})$.

- H_{1tempo}: A codificação de criação automática é mais eficiente que a codificação manual.

$(\mu_{tempoCodificaçãoManual} > \mu_{tempoCodificaçãoAutomática})$.

HIPÓTESE 2

- H_{0erros}: A codificação de criação automática e manual tem a mesma eficácia.

$(\mu_{grauErrosCodificaçãoManual} = \mu_{grauErrosCodificaçãoAutomática})$.

- H_{1erros}: A codificação de criação automática é mais eficaz que a codificação manual.

($\mu_{grauErrosCdificaçãoManual} > \mu_{grauErrosCodificaçãoAutomática}$).

Para ambas as hipóteses, a $H_{0,}$ é a hipótese que se deseja rejeitar. A hipótese alternativa, $H_{1,}$ a que se deseja não rejeitar. Para averiguar quais das hipóteses são rejeitadas, serão consideradas as variáveis dependentes e independentes que se encontram na Figura 24.

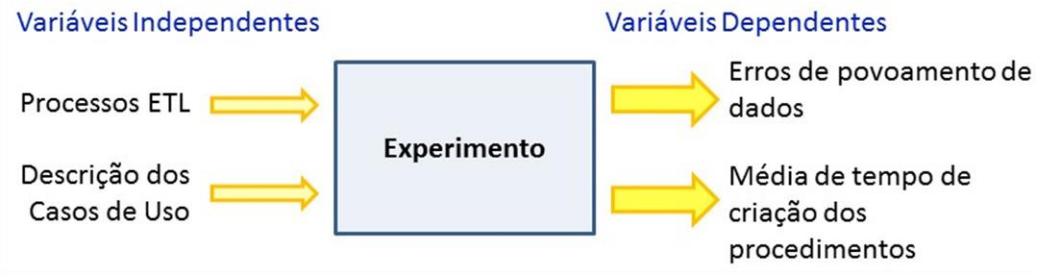

Figura 24: Variáveis Dependentes e Independentes do Experimento

A) VARIÁVEIS INDEPENDENTES

A seguir, são descritas as variáveis independentes do experimento. As variáveis independentes do experimento são a descrição dos Casos de Uso e Processos ETL para carga de dados históricos. Para os Casos de Uso, serão consideradas as situações da prática e os tipos de tratamento histórico dos dados efetivamente usados pelos programadores selecionados, relatados como os utilizados no mercado. São os tipos 1 e 2, abordados a seguir.

DESCRIÇÃO DE CASOS DE USO UTILIZADOS NO EXPERIMENTO

I) UC01 – CLIENTES

O objetivo, neste Caso de Uso, será gerar procedimento para realizar carga da Área de *Staging*[8], de uma tabela de Clientes, para uma Dimensão (SANTOS; BELO, 2011), nomeada Cliente. Para este caso, a dimensão não terá como propósito o armazenamento histórico dos dados. Ou seja, todos os atributos terão o comportamento do Tipo 1.

O Quadro 4 apresenta as características da dimensão Cliente no ambiente de DW.

Quadro 4: Característica da dimensão dw.DIM_Cliente e seus atributos quanto ao armazenamento de histórico.

Nome da dimensão	dw.DIM_Cliente
Tipo de tratamento de histórico	Tipo 1

[8] Área intermediária, entre o ambiente transacional e o DW, onde os dados precisam ser preparados para povoar, em seguida, o DW (SANTOS; BELO, 2011).

Atributo	**Tipo**	**Papel do Atributo**
id_cliente		Surrogate Key
cod_cliente		Chave natural
Cpf	1	
nome_cliente	1	
data_nascimento	1	
Endereco	1	
Bairro	1	
Cidade	1	

II) UC02 – PRODUTOS

Para este Caso de Uso, o objetivo será gerar procedimento para realizar cargas da Área de *Staging*, de uma tabela de Produtos, para a Dimensão Produto. Neste momento, a dimensão terá armazenamento histórico, Tipo 2, para alguns atributos. Os demais atributos serão do Tipo 1.

O Quadro 5 apresenta as características da dimensão Produto no ambiente de DW.

Quadro 5: Característica da dimensão dw.DIM_Produto e seus atributos quanto ao armazenamento de histórico.

Nome da dimensão	dw.DIM_Produto	
Tipo de tratamento de histórico	Tipo 1 e 2	
Atributo	**Tipo de Histórico**	**Papel do Atributo**
id_produto		Surrogate Key
codigo_produto		Chave Natural
codigo_barra	1	
Descricao	2	
Linha	2	
Valor	2	
dt_inicio		Data Inicial
dt_fim		Data Final

fl_corrente		Flag Corrente

Para o tratamento de atributos Tipo 2, em dimensões, novos atributos são utilizados para o armazenamento histórico. São eles: a data inicial, que representa a data em que o registro foi gravado; a data final, que representa a data em que o registro deixou de ser corrente; e por fim, o atributo que identifica se é um registro atual ou não. Estes são apresentados respectivamente no Quadro 5, com os nomes de: dt_inicio; dt_fim e fl_corrente.

PROCESSOS ETL

Os processos ETL, utilizados neste trabalho, possuem dois tipos de tratamentos para a realização do experimento:

1) Codificação Manual: criação manual dos procedimentos ETL, em SQL, para carga de dados definida nos dois Casos de Uso já apresentados anteriormente.
2) Codificação Automática: geração de código SQL para os mesmos Casos de Uso, utilizando a ferramenta FGCod, que será descrita mais adiante na seção de Instrumentação.

4.1.2.2 Seleção de Participantes

O processo de seleção dos participantes ocorreu por conveniência. O colaborador escolhido foi a Qualycred (www.qualycred.com), uma empresa que fornece consultoria em soluções de *Business Intelligence* para Indústria. A mesma forneceu, para a execução do experimento, oito programadores com quatro anos de experiência em outras áreas e um ano de experiência, atuando especificamente com ETL para DW, no SGBD SQL SERVER.

4.1.2.3 Projeto do Experimento

Projetou-se o experimento num contexto pareado em que um grupo avaliará ambas as abordagens: Codificação para criação Manual e Automática. Para entendimento da codificação a ser feita, foram elaborados dois Casos de Uso (vistos na seção Formulação de Hipóteses), os quais serão apresentados de forma bem detalhada aos programadores. O experimento será separado em dois momentos, sorteando 4 programadores para iniciar com a

opção 1 e os restantes com a opção 2, são elas: 1º) Codificação feita para as regras apresentadas no Caso de Uso 01, com a Geração do Código Manual e, logo após, a Geração do Código Automático, ambas seguidas da execução do procedimento produzido; 2º) Codificação feita para regras apresentadas no Caso de Uso 02, com a Geração do Código Automático e, logo após, a Geração do Código Manual, ambas seguidas da execução do procedimento. Desta forma, será favorecida a aleatoriedade, não priorizando o aprendizado manual ou automático. Os programadores poderão copiar e colar seus códigos já escritos na indústria.

4.1.2.4 Instrumentação

O processo de instrumentação se deu inicialmente com a configuração do ambiente para o experimento e planejamento de coleta dos dados. Este será realizado em um laboratório de informática da Universidade Federal de Sergipe – UFS. Para que os participantes do experimento tenham as mesmas condições de trabalho, os computadores serão ajustados a fim de possuírem as mesmas configurações.

Seguem, abaixo, a tecnologia, as ferramentas instaladas e artefatos.

4.1.2.4.1 SQL Server 2008

O Sistema Gerenciador de Banco de Dados SQL SERVER (SQLSERVER, 2008), na versão 2008, serviu como base para o armazenamento dos metadados identificados, e, consequentemente, foi utilizado para armazenar os metadados da FGCod descrita a seguir.

4.1.2.4.2 Ferramenta de Geração Automática de Código Sql para ETL (FGCod)

A FGCod está bem descrita na seção 3.3, deste trabalho. A versão a ser utilizada no experimento, devido aos participantes, gera códigos *SQL*, na linguagem T-SQL (*Transact-SQL*), envolvendo comportamentos do Tipo 1 e 2 para o tratamento de histórico dos atributos em dimensões.

4.1.2.4.3 Ambiente Criado e Artefatos Produzidos

Foi definido e criado um ambiente de *Data Warehouse* com os esquemas dimensionais e áreas de *Staging* para cada programador.

4.2 Operação do Experimento

4.2.1 Preparação

A seguir, são enumeradas as etapas de preparação para a execução do experimento.

1) *Alocação de programadores a Casos de Uso*: foi apresentado, a cada programador, um documento impresso, contendo uma descrição detalhada dos Casos de Uso que seriam utilizados por eles, em caso de eventuais dúvidas;

2) *Revisão de conceitos básicos sobre rotinas de povoamento para o grupo de programadores*: nessa fase, foi realizada uma revisão sobre as rotinas de povoamento para ambientes de DW com os programadores selecionados;

3) *Treinamento da ferramenta FGCod*: pela facilidade de uso, foi realizado um treinamento de 30 minutos com os programadores, ministrado por uma pessoa não envolvida com o experimento, a fim de que eles pudessem se familiarizar com a ferramenta de geração automática de código.

Em resumo, todos os computadores foram preparados, com as mesmas configurações, para que os programadores estivessem sobre as mesmas condições de trabalho. Além disso, foi apresentado, a cada programador, um documento impresso, contendo uma descrição detalhada dos Casos de Uso que seriam utilizados por eles, em caso de eventuais dúvidas.

4.2.2 Execução

Ao final das etapas anteriores, deu-se início ao experimento, com a realização da geração de código para o *UC01 – Clientes*, primeiramente de forma manual, e, posteriormente, utilizando a ferramenta para a geração automática de código. Feito isto, houve a execução dos procedimentos gerados para povoar o *DW*, a partir dos dados provenientes da área de *Staging*. A codificação manual serviu para fazer a comparação com as rotinas geradas pela ferramenta. Paralelamente (subgrupo 2), foi feita a geração de código para *UC02* –

Produtos, seguindo a mesma estratégia do Caso de Uso anterior, trocando apenas a ordem de geração de código, que para este foi feito, primeiramente, de forma automática, e, posteriormente, de forma manual. Este procedimento foi realizado duas vezes, devido ao sorteio planejado.

A avaliação da ferramenta, ao final do experimento, por parte dos profissionais, foi positiva, visto que os mesmos comentaram que a usabilidade da ferramenta contribui para a redução do tempo no desenvolvimento dos procedimentos.

4.2.2.1 Como os dados foram coletados

Foi calculado o tempo gasto por cada programador, para codificação manual e para codificação Automática, de cada Caso de Uso, levando em consideração o tempo para implementação e execução. Ao término de todo o experimento, foram armazenados os procedimentos manuais e automáticos para análise de erros.

Os procedimentos mantidos pelos programadores foram avaliados individualmente, comparando-os a um procedimento padrão, considerado como um "modelo" a ser seguido, o qual já possui uma estrutura bem testada e aprovada por empresas. Para execução de cada procedimento, foi analisada a qualidade dos dados carregados para o DW, por um profissional conhecedor do negócio, identificando possíveis inconsistências existentes. Após a detecção de inconsistências, foram atribuídos pesos às mesmas, a depender do grau de criticidade apresentado.

Os resultados desses dados coletados serão apresentados na seção 4.3.

4.2.3 Validação dos Dados

Para realização do experimento, foi considerado um fator, implementação do procedimento *SQL*, e dois tratamentos, implementação de forma manual e implementação de forma automática, utilizando a ferramenta FGCod. Diante deste contexto, foram computadas as médias do tempo de implementação e médias de grau de criticidade de erros encontrados nos dados povoados no DW após execução dos procedimentos.

Como auxilio para análise, interpretação e validação, foram utilizados três tipos de testes estatísticos, Teste Shapiro-Wilk, Teste T e Teste de Wilcoxon. O Teste Shapiro-Wilk

foi utilizado para verificar normalidade das amostras. O Teste T foi utilizado para comparar a média de duas amostras pareadas (WOHLIN et al., 2000) e, por fim, o Teste de Wilcoxon para comparar as médias das amostras pareadas que não obtiveram normalidade nos dados, verificando a magnitude da diferença (WOHLIN et al., 2000).

Todos os testes estatísticos foram feitos utilizando a Ferramenta SPSS – IBM (SPSS, 2014).

4.3 Resultados

4.3.1 Análise e Interpretação de Dados

Para responder às questões, foram analisadas as seguintes variáveis dependentes: a) o tempo para criação de cada procedimento e; b) os erros encontrados no povoamento.

Tempo de Criação dos Procedimentos

Para responder à Questão de Pesquisa 01, os resultados relacionados ao tempo de implementação dos procedimentos por cada participante são apresentados na Tabela 1.

Para o Caso de Uso 01 (UC01), os resultados mostram que a média de tempo dos programadores para implementarem o procedimento de forma manual foi de 38,5 minutos e de forma automática 9,12 minutos. No Caso de Uso 02 (UC02), para implementação manual, os programadores obtiveram uma média de 39,5 minutos e, de forma automática, 8,5 minutos.

Tabela 1: Tempos de implementações individuais por programador.

	UC01		UC02	
	Manual (minutos)	**Automática (minutos)**	**Manual (minutos)**	**Automática (minutos)**
Programador1	29	7	26	7
Programador 2	39	10	30	4
Programador 3	52	10	54	15
Programador 4	29	13	26	6
Programador 5	35	7	43	12
Programador 6	35	9	54	9

Programador 7	61	10	54	10
Programador 8	26	7	29	5

Esses resultados sugerem que a criação de procedimentos de forma automática possui, em média, menor tempo de implementação, quando comparado com a criação dos mesmos procedimentos de forma manual por programadores com experiência na área. Com isso, a partir desta análise prévia dos dados, supõe-se que a resposta para a Questão 01 de Pesquisa seria "sim", que a geração automática de código pode reduzir o tempo dos programadores no processo de povoamento de um DW, já que o desenvolvimento feito de forma automática obteve uma diferença de aproximadamente 30 minutos, para ambos os Casos de Uso. Mas não é possível fazer tal afirmação sem evidências estatísticas suficientemente conclusivas.

Para isto, primeiramente, definimos um nível de significância de 0.05 em todo o experimento e foi aplicado o Teste de Shapiro-Wilk, para análise da distribuição normal das amostras (Tabela 2 e Tabela 3), UC01. Em ambas amostras, implementação manual e automática, os valores da variável Sig., 0.148 e 0.128, leia-se *p-value*, associados ao teste de shapiro-wilk, são maiores que o nível de significância adotado. Assume-se que a distribuição dos dados, para ambas as implementações, é normal.

Tabela 2: Teste de Shapiro-Wilk em relação ao tempo de implementação manual para o Caso de Uso 01. (Fonte: Ferramenta SPSS - IBM).

	Shapiro-Wilk		
	Statistic	df	Sig.
TempoUC01Manual	,869	8	,148

Tabela 3: Teste de Shapiro-Wilk em relação ao tempo de implementação automática para o Caso de Uso 01. (Fonte: Ferramenta SPSS - IBM).

	Shapiro-Wilk		
	Statistic	df	Sig.
TempoUC01Automatico	,863	8	,128

Sendo assim, o Teste de hipótese aplicado neste contexto será o Teste T, caracterizado como paramétrico, para amostras emparelhadas. Para isso, verificou-se que o Sig. de 0.000, (Tabela 4), é fortemente menor que o nível de significância adotado. Desta forma, confirmou-se a evidência de diferença entre as médias de 29,125 (Tabela 4), para o UC01.

Posteriormente, foi aplicado o Teste de Shapiro-Wilk, para análise da distribuição normal das amostras, no UC02. O valor da variável Sig., para implementação manual, foi 0.023, e 0.779, na implementação automática. Um destes valores é menor que o nível de significância adotado. Assume-se que a distribuição dos dados, para implementação manual, não é normal.

Tabela 4: Teste T em relação ao tempo de implementação. (Fonte: Ferramenta SPSS - IBM).

		Paired Differences					T	Df	**Sig**
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 1	TempoUC01Manual TempoUC01 Automatico	**29,125**	11,837	4,185	19,229	39,021	6,96	7	**,000**

Desta forma, no UC02, será utilizado o Teste de Wilcoxon, para o Teste de hipótese com amostras emparelhadas que não obtiveram normalidade dos dados. Para isso, verificou-se um Sig. de 0.012 , menor que o nível de significância de 0.05. Com isso, confirmou-se a evidência de diferença entre as médias, para o UC02.

Diante do apresentado, confirmou-se a evidência de diferença entre as médias, para o UC01 e UC02. O tempo médio de implementação é significativamente diferente, ou seja, a hipótese (H_0), de que a codificação automática e manual têm a mesma eficiência foi rejeitada, pois o teste de significância é < 0.05 para ambos Casos de Uso. Consequentemente, não é rejeitada a hipótese alternativa de que a codificação automática é mais eficiente que a codificação manual.

Erros de Povoamento dos Dados

Para responder à Questão de Pesquisa 02, foram comparados os erros, contabilizados levando em conta o esforço através da matriz GUT (Gravidade, Urgência e Tendência) (MARSHALL et al., 2011) (Tabela 5), encontrados após execução dos Procedimentos SQL criados de forma manual e automática, para dois Casos de Uso diferentes. Os resultados relacionados ao grau de criticidade de erros dos procedimentos por cada participante são apresentados na Tabela 6. Estes erros foram encontrados com abordagem *blind*, com o auxílio de um usuário de negócio com conhecimento do sistema OLTP, o qual não estava a par do experimento.

Na matriz GUT, Tabela 5, são definidos os problemas que foram encontrados nos procedimentos gerados no experimento. Além disso, é estabelecido o valor para os índices de gravidade, urgência e tendência de erros para povoamento de dados em um *DW*.

O primeiro problema detectado, Tabela 5, foi a eliminação de antigos registros e inserção dos mesmos para os dados mais atuais provenientes da área de *Staging*, para atributos do tipo 1 (sem necessidade de guardar histórico). Esse acontecimento se torna um problema por violar os valores das *Surrogate Keys* das dimensões, visto que, se os registros já estiverem vinculados à tabela de fatos (indicadores de desempenho do negócio), estes não conseguirão relacionar-se com os registros da dimensão, por haver alterações nos valores das chaves estrangeiras.

Tabela 5: Matriz GUT.

	Problemas	**G**	**U**	**T**	**Grau Crítico**
1	Eliminação e inserção de registros (com comportamento tipo 1) já encontrados na dimensão.	3	3	3	27
2	Falta de atualização para alguns atributos com comportamento do tipo 1.	3	3	4	36
3	Erro na atualização do atributo que representa a data final para uma dimensão com atributos tipo 2.	3	2	2	12
4	Não atualização de dados.	5	5	5	125
5	Duplicação de dados.	4	4	4	64

No segundo caso, linha 2 da Tabela 5, ocorre a falha na atualização para atributos com comportamento do tipo 1. No terceiro problema, é identificado o erro para o tratamento de histórico do atributo de data final, com comportamento tipo 2 (armazenamento de histórico). Esse erro infringe a regra de atualização e armazenamento correto da data final para os registros que não são mais correntes ou válidos atualmente.

O pior caso envolve o problema 4. A não atualização de dados com o tratamento correto de histórico na dimensão, é uma falta grave e deve ser diagnosticada o quanto antes no processo de povoamento.

Por fim, a duplicação de dados, problema 5, está atrelado à inconsistência e redundância em um ambiente de DW. Esta redundância leva a altos custos de armazenamento e acesso aos dados.

Para o Caso de Uso 01 (UC01), os resultados mostram que a média de grau de criticidade de erros dos programadores, para implementarem o procedimento de forma manual, foi 26,88, e, de forma automática, 0 (zero), ou seja, ao serem executados os procedimentos criados por cada programador através da ferramenta FGCod, não foram encontrados erros. No Caso de Uso 02 (UC02), para implementação manual, foi obtida uma média de grau de criticidade de erros de 83,38. Para geração automática, a média também foi 0 (zero).

Tabela 6: Grau de criticidade de erros contabilizados para cada programador.

	UC01 – Clientes			**UC02 – Produtos**		
		Total de Grau Crítico			**Total de Grau Crítico**	
	Problemas	Manual	Automático	**Problemas**	Manual	Automático
Programador1	-	0	0	-	0	0
Programador 2	-	0	0	2	12	0
Programador 3	-	0	0	2; 3; 4	173	0
Programador 4	1	27	0	2; 3	48	0
Programador 5	2	36	0	3	12	0
Programador 6	1	27	0	2; 3; 4	173	0
Programador 7	-	0	0	2; 3; 4	173	0
Programador 8	4	125	0	3; 5	76	0

Pela constância da ausência de erros na geração automática e consequente não normalidade dos dados, já que as diferenças destes erros não se distribuem normalmente, foi aplicado o Teste de Wilcoxon como Teste de Hipótese. Esse se caracteriza como não paramétrico, para amostras emparelhadas, levando em consideração que a amostra tem um comportamento contínuo e simétrico. O teste, além de comparar a diferença das amostras, também verifica a magnitude dessa diferença.

Com aplicação do Teste de Wilcoxon, para o UC01, verificou-se que o Sig. de 0.066 (Tabela 7) é maior que o nível de significância de 0.05, ou seja, não é possível afirmar que as médias são estatisticamente diferentes. Neste UC01, para metade dos procedimentos desenvolvidos de forma manual, não foram apresentados erros. Acredita-se que este resultado se deu pelo fato de ser um Caso de Uso simples, que apresentava poucas restrições, ficando mais perceptível para o programador detectar e implementar todas as especificações. Diferentemente do UC02, o qual apresenta maior grau de dificuldade para implementação.

Tabela 7: Teste Wilcoxon em relação aos erros para UC01(Fonte: Ferramenta SPSS - IBM).

	ErrosUC01Automatico - ErrosUC01Manual
Z	-1,841
Asymp. Sig. (2-tailed)	**,066**

Para o UC02, também foi aplicado o Teste de Wilcoxon. Verificou-se que o valor de Sig. é 0.017, menor que o nível de significância adotado. Assim, confirmou-se a evidência de diferença entre as médias de grau de criticidade de erros, para o UC02.

De forma geral, diante dos resultados, confirmam-se evidências de que os erros de implementação de forma manual e automática são significativamente diferentes, para casos de usos mais complexos. Desta forma, a hipótese (H_0) de que a codificação automática e manual têm a mesma eficácia foi rejeitada apenas para o UC02.

4.3.2 Ameaças à validade

Embora os resultados do experimento tenham se mostrado satisfatórios, o mesmo apresenta ameaças à sua validade que não podem ser desconsideradas.

Ameaças à validade interna: Como há coincidências nas declarações e funcionalidades dos procedimentos a serem criados para os dois casos de uso, o código produzido no UC01 pode ter sido utilizado como base para criação manual do procedimento no UC02 e vice-versa. Isso pode ter ajudado a diminuir o tempo de construção e erros para o UC02, mais complexo. No entanto, a ferramenta mostrou-se mais eficaz também para este caso. De fato, são casos como este que pretendemos beneficiar, dado que na prática, conforme relatado pela empresa colaboradora, são os casos que apresentam mais problemas.

Ainda que os participantes tenham sido treinados a utilizarem a ferramenta, os mesmos não a utilizam diariamente. Essa falta de contato constante com a mesma pode ter afetado os resultados, os quais poderiam ser ainda melhores, pró-ferramenta. O treinamento da ferramenta foi realizado logo no início do experimento, considerando um fenômeno estudado pela psicologia denominado *Demand Characterization*, o qual considera que um artefato experimental pode ter uma interpretação, pelos participantes, do propósito do experimento. Isto pode levar à mudança de comportamento inconsciente, para se adaptar a esta interpretação (ORNE, 2009). De acordo com este conceito, este treinamento poderia ter

prejudicado o andamento do experimento, mas, para mitigar este fator, pode-se dizer que foram utilizadas pelo menos duas abordagens diferentes: *The More The Merrier* e *Unobtrusive Manipulations and Measures* (ORNE, 2009). Respectivamente, na primeira, para evitar o viés, com um único experimentador, o experimento contou com mais um pesquisador para conduzir o experimento e um instrutor para a ferramenta, não envolvido com a pesquisa. A segunda nos norteou a não informar quais fatores e métricas seriam avaliados, de modo que os participantes não tivessem pistas sobre a hipótese de pesquisa.

Ameaças à validade externa: o baixo número de participantes é uma ameaça, visto que pode influenciar os resultados do experimento. Mas, em Engenharia de Software, pelo custo de experimentos controlados com programadores que possuem experiência com processos do mercado, há uma exponencial dificuldade em conseguir disponibilidade empresarial para liberação destes profissionais. Para este experimento, foi proposta a utilização, privilegiada e cobiçada pela comunidade científica de Engenharia de Software, de pessoas do mercado, mitigando esta ameaça com uma cota estratificada de programadores com experiência na área de ETL para BI. Assim, assume-se que houve uma razoável representatividade perante a população de desenvolvedores de procedimentos ETL.

Ameaças à validade de construção: As Especificações dos Casos de Uso podem não ter ficado muito claras ao entendimento de algum programador. Esta ameaça foi mitigada com a leitura prévia e análise do entendimento de 3 programadores ETL.

CAPÍTULO 5

5. EXPERIMENTO II

Este experimento se refere à avaliação do módulo da FGCod para manutenção automatizada de procedimentos SQL em ambientes de BI.

5.1 Definição e Planejamento do Experimento

Nesta seção, o conteúdo é apresentado como um processo experimental. O mesmo segue as diretrizes de Wohlin *et al.* Em (WOHLIN et al., 2000). Esta seção irá focar na definição do objetivo e no planejamento do experimento. A Operação e os Resultados não serão descritos e obtidos neste momento.

5.1.1 Definição do Objetivo

O objetivo deste experimento foi avaliar, por meio de um experimento controlado, o uso da geração automática de código na manutenção de rotinas de povoamento em um ambiente de *Data Warehouse*, através de uma ferramenta de Desenvolvimento Rápido de Aplicações (RAD), a FGCod.

O experimento teve como alvo programadores de processos ETL para ambientes de *Business Intelligence,* com alguma experiência no mercado. O objetivo foi formalizado utilizando o modelo GQM proposto por Basili (BASILI; WEISS, 1984): **Analisar** o uso de uma ferramenta de geração de código automático, na manutenção de procedimentos SQL, para povoar um ambiente DW, **com a finalidade de** avaliar se a utilização de uma ferramenta

de geração automática de código pode substituir a codificação da manutenção manual, **com respeito à** eficiência e eficácia do processo de geração automática de código, **do ponto de vista de** programadores e gestores de suporte à decisão, **no contexto de** programadores de uma empresa de *BI*.

5.1.2 Planejamento

5.1.2.1 Formulação de Hipóteses

As questões de pesquisa para o experimento que precisavam ser respondidas eram as seguintes: 1ª) A geração automática de código pode reduzir o tempo dos programadores no processo de manutenção de procedimentos para povoamento de um DW?; 2ª) A geração automática de código pode reduzir ou eliminar erros na codificação para manutenção de procedimentos e nos dados povoados em um DW?

Para avaliar estas questões, foram utilizadas duas métricas como variáveis dependentes: 1ª) Média de tempo, para Codificação da manutenção Manual e para Codificação da manutenção Automática; 2ª) Média de grau de criticidade de erros de povoamento, calculada considerando: (a) Registros carregados incorretamente para o DW: (1) Erro e falta de tratamento histórico para atributos das dimensões; (2) Registros duplicados; (3) Não atualização dos dados. (b) Utilização de matriz GUT (sigla para Gravidade, Urgência e Tendência) (MARSHALL et al., 2011) para gerenciamento de problemas através dos erros citados no item (a), bem como cálculo do grau de criticidade.

Tendo os objetivos e métricas definidas, foram consideradas as hipóteses:

HIPÓTESE 1

- H_{0tempo}: A codificação para manutenção automática e manual tem a mesma eficiência.

$(\mu_{tempoCodificaçãoManual} = \mu_{tempoCodificaçãoAutomática})$.

- H_{1tempo}: A codificação para manutenção automática é mais eficiente que a codificação manual. $(\mu_{tempoCodificaçãoManual} > \mu_{tempoCodificaçãoAutomática})$.

HIPÓTESE 2

- H_{0erros}: A codificação para manutenção automática e manual tem a mesma eficácia.

$(\mu_{grauErrosCodificaçãoManual} = \mu_{grauErrosCodificaçãoAutomática})$.

- H_{1erros}: A codificação para manutenção automática é mais eficaz que a codificação manual. ($\mu_{grauErrosCdificaçãoManual} > \mu_{grauErrosCodificaçãoAutomática}$).

Para ambas as hipóteses, a $H_{0,}$ foi a hipótese que se desejava rejeitar. A hipótese alternativa, $H_{1,}$ a que se desejava não rejeitar. Para averiguar quais das hipóteses seriam rejeitadas, foram consideradas as variáveis dependentes e independentes que se encontram na Figura 25.

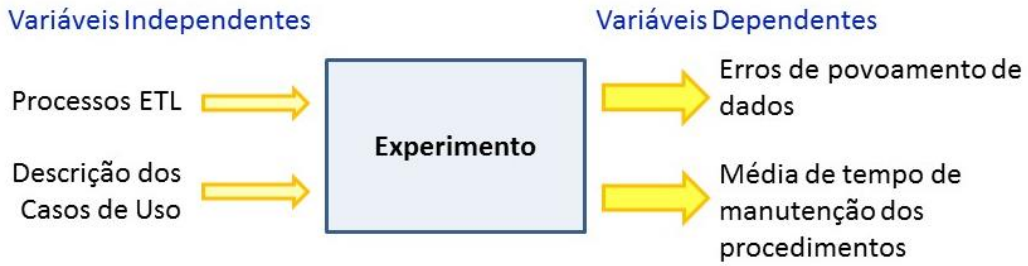

Figura 25: Variáveis Dependentes e Independentes do Experimento II

A) VARIÁVEIS INDEPENDENTES

As variáveis independentes do experimento são a descrição dos Casos de Uso e Processos ETL para carga de dados históricos. Para os Casos de Uso, foram consideradas as situações da prática e os tipos de tratamento histórico dos dados efetivamente usados pelos programadores selecionados, relatados como os utilizados no mercado. São os tipos 1 e 2, abordados a seguir.

DESCRIÇÃO DE CASOS DE USO UTILIZADOS NO EXPERIMENTO

I) UC01 – MANTER CLIENTES

O objetivo, neste Caso de Uso, foi dar manutenção em um procedimento que realiza carga de uma tabela de Clientes, para uma Dimensão do DW, nomeada DIM_Cliente. Inicialmente o procedimento só tratava de atributos que não armazenavam histórico, ou seja, do Tipo 1. Após manutenção, o procedimento atendeu ao armazenamento de histórico Tipo 2, para alguns atributos. Alguns atributos da dimensão não armazenaram histórico, ou seja, foram do tipo 1, outros atributos armazenarão histórico de acordo com os critérios apresentados no armazenamento de histórico do tipo 2. O QUADRO 6 ilustra qual o tipo de armazenamento de histórico de cada atributo, antes da manutenção, já o Quadro 7, apresenta o tipo de armazenamento de histórico após manutenção.

Quadro 6: Característica da dimensão dw.DIM_Cliente e seus atributos quanto ao armazenamento de histórico – antes da manutenção.

Nome da dimensão	dw.DIM_Cliente	
Tipo de tratamento de histórico	tipo 1	
Atributo	**Tipo**	**Papel do Atributo**
data_carga		Data de Carregamento
cod_cliente		Chave natural
cpf	1	
nome_cliente	1	
data_nascimento	1	
endereco	1	
bairro	1	
cidade	1	
estado	1	
dt_inicio		Data Inicial
dt_fim		Data Final
fl_corrente		Flag Corrente

Quadro 7: Característica da dimensão dw.DIM_Cliente e seus atributos quanto ao armazenamento de histórico – após manutenção.

Nome da dimensão	dw.DIM_Cliente	
Tipo de tratamento de histórico	tipo 1 e 2	
Atributo	**Tipo**	**Papel do Atributo**
data_carga		Data de Carregamento
cod_cliente		Chave natural
cpf	1	
nome_cliente	1	
data_nascimento	1	
endereco	2	
bairro	2	
cidade	2	
telefone	1	
dt_inicio		Data Inicial
dt_fim		Data Final
fl_corrente		Flag Corrente

As especificações do que foi feito na manutenção estão descritas no quadro abaixo, Quadro 8.

Quadro 8: Especificação do Caso de Uso 01 – Manter Clientes

[UC 01] Manter Atributos Cliente	
DESCRIÇÃO:	Alteração e Inclusão de Atributos no procedimento "CarregaCliente".
Ator:	DBA (Administrador de banco de dados) e Programadores
Prioridade:	Essencial
Entradas e Pré-condições:	Ter o Procedimento já criado para carregar dados dos Clientes da área de *Staging* para o DW.
Saídas e pós-condições:	- Atributos referentes à Endereço (endereço, bairro e cidade) alterados o Tipo de histórico, no procedimento. - Atributo referente à Telefone Incluso no procedimento. Atributo referente à Estado Excluído do procedimento.
Fluxo básico:	1. O procedimento "CarregaCliente" necessita de manutenção, para suprir necessidades de Negócio. 2. Será necessário executar os seguintes Subfluxos: a. Subfluxo Alterar b. Subfluxo Incluir c. Subfluxo Excluir
Subfluxo Alterar	1. Ao ser feita carga de dados da área de Staging (tabela auxiliar) para a Dimensão Cliente **(dw.DIM_Cliente)** será necessário armazenar histórico do endereço do Cliente. 2. O Tipo de histórico necessário é o Tipo 2. 3. Os Seguintes atributos devem ser alterados para armazenamento do Tipo 2: - "endereco" - "bairro" - "cidade"
Subfluxo Incluir	1. Uma nova Coluna foi adicionada para Clientes. 2. A inclusão de um Atributo para armazenar telefone, foi incluso na Tabela de "staging.TB_Aux_Cliente" e "dw.DIM_Cliente". 3. O atributo referente ao telefone, tanto na "staging.TB_Aux_Cliente" como na "dw.DIM_Cliente" é "telefone". 4. O Atributo "telefone" não tem armazenamento de histórico, ou seja, é do Tipo 1.
Subfluxo Excluir	1. Uma nova Coluna foi excluída para Clientes.

	2. A exclusão do Atributo para armazenar o Estado do Cliente, foi excluído da Tabela de **"staging.TB_Aux_Cliente"** e **"dw.DIM_Cliente"**. 3. O atributo referente à Estado do Cliente, tanto na **"staging.TB_Aux_Cliente"** como na **"dw.DIM_Cliente"** era "estado".

II) UC02 – MANTER PRODUTOS

O objetivo, neste Caso de Uso, foi dar manutenção em um procedimento que realiza carga da tabela de Produtos, para uma Dimensão do DW, nomeada Dim_Produto. A dimensão teve armazenamento histórico, Tipo 2, para alguns atributos. Os demais atributos foram do Tipo 1.

O Quadro 9 ilustra qual o tipo de armazenamento de histórico de cada atributo, antes da manutenção, já o

Quadro 10, apresenta o tipo de armazenamento de histórico após manutenção.

Quadro 9: Característica da dimensão dw.DIM_Produto e seus atributos quanto ao armazenamento de histórico – antes da manutenção.

Nome da dimensão	dw.DIM_Produto	
Tipo de tratamento de histórico	Tipo 1 e 2	
Atributo	**Tipo de Histórico**	**Papel do Atributo**
id_produto		Surrogate Key
codigo_produto		Chave Natural
codigo_barra	1	
Descricao	2	
Linha	2	
Valor	2	
composicao	2	
dt_inicio		Data Inicial
dt_fim		Data Final
fl_corrente		Flag Corrente

Quadro 10: Característica da dimensão dw.DIM_Produto e seus atributos quanto ao armazenamento de histórico – após manutenção.

Nome da dimensão	dw.DIM_Produto	
Tipo de tratamento de histórico	Tipo 1 e 2	
Atributo	**Tipo de Histórico**	**Papel do Atributo**
id_produto		Surrogate Key

codigo_produto		Chave Natural
codigo_barra	1	
descricao	2	
Linha	1	
Valor	2	
fornecedor	2	
dt_inicio		Data Inicial
dt_fim		Data Final
fl_corrente		Flag Corrente

As especificações do que foi feito na manutenção estão descritas no quadro abaixo, Quadro 11.

Quadro 11: Especificação do Caso de Uso 02 – Manter Produtos

[UC 02] Manter Atributos Produto	
DESCRIÇÃO:	Alteração, Inclusão e Exclusão de Atributos no procedimento "CarregaProduto".
Ator:	DBA (Administrador de banco de dados) e Programadores
Prioridade:	Essencial

Entradas e Pré-condições:	Ter o Procedimento já criado para carregar dados dos Produtos da área de *Staging* para o DW.
Saídas e pós-condições:	- Atributo referente a Linha do Produto alterado o Tipo de histórico. - Atributo referente à Fornecedor Incluso. - Atributo referente à Composição do Produto Excluído.
Fluxo básico:	1. O procedimento "CarregaProduto" necessita de manutenção, para suprir necessidades de Negócio. 2. Será necessário executar os seguintes Subfluxos: a. Subfluxo Alterar b. Subfluxo Incluir d. Subfluxo Excluir
Subfluxo Alterar	1. Ao ser feita carga de dados da área de Staging (**staging.TB_Aux_Produto**) para a Dimensão Produto (**dw.DIM_Produto)** não será mais necessário armazenar histórico para a Linha do Produto. 2. O Tipo de histórico necessário é o Tipo 1. 3. O Seguinte atributo deve ser alterado para armazenamento do Tipo 1: - "linha"

Subfluxo Incluir	1. Uma nova Coluna foi adicionada para Produtos. 2. A inclusão de um Atributo para armazenar Fornecedor do Produto, foi incluído na Tabela de **"staging.TB_Aux_ Produto" e "dw.DIM_ Produto".** 3. O atributo referente ao Fornecedor, tanto na **"staging.TB_Aux_Produto"** como na **"dw.DIM_Produto"** é "fornecedor". 4. O Atributo "fornecedor" terá armazenamento de histórico, Tipo 2.
Subfluxo Excluir	1. Uma nova Coluna foi excluída para Produtos. 2. A exclusão do Atributo para armazenar a Composição do Produto, foi excluído da Tabela de **"staging.TB_Aux_Produto" e "dw.DIM_Produto".** 3. O atributo referente à Composição do Produto, tanto na **"staging.TB_Aux_Produto"** como na **"dw.DIM_Produto"** era "composicao".

PROCESSOS ETL

Os processos ETL, utilizados neste trabalho, possuem dois tipos de tratamentos para a realização do experimento:

1) Codificação Manual: manutenção manual dos procedimentos ETL, em SQL, para carga de dados definida nos Casos de Uso que serão definidos.

2) Codificação Automática: manutenção de código SQL para os mesmos Casos de Uso, utilizando a ferramenta a FGCod.

5.1.2.2 Seleção de Participantes

O processo de seleção dos participantes ocorreu por conveniência. O colaborador escolhido foi a Qualycred (www.qualycred.com), uma empresa que fornece consultoria em soluções de *Business Intelligence* para Indústria. A mesma forneceu 8 programadores com quatro anos de experiência em outras áreas e dois anos de experiência, atuando especificamente com ETL para DW, no SGBD SQL SERVER.

5.1.2.3 Projeto do Experimento

Projetou-se o experimento num contexto pareado, em que um grupo avaliou ambas as abordagens: Codificação para manutenção Manual e Automática. Para entendimento da manutenção feita, foram modificados Casos de Uso, os quais foram apresentados de forma bem detalhada aos programadores. O experimento foi separado em dois momentos, sorteando metade da quantidade de programadores para iniciar com a opção 1 e os restantes com a opção 2, seguida da inversão das opções entre os grupos, na sequência. As duas opções foram: 1º) Codificação da manutenção feita para as regras alteradas no primeiro Caso de Uso, com a Geração do Código Manual e, logo após, a Geração do Código Automático para o segundo Caso de Uso, ambas seguidas da execução do procedimento produzido; 2º) Codificação feita para regras alteradas no segundo Caso de Uso, com a Geração do Código Manual e, logo após, a Geração do Código Automático para o primeiro Caso de Uso, ambas seguidas da execução do procedimento. Desta forma, foi favorecida a aleatoriedade, não priorizando o aprendizado manual ou automático. Os programadores puderam copiar e colar seus códigos já escritos na indústria.

5.1.2.4 Instrumentação

O processo de instrumentação se deu inicialmente com a configuração do ambiente para o experimento e planejamento de coleta dos dados. Este foi realizado em um laboratório de informática da Universidade Federal de Sergipe – UFS. Foi utilizado um ambiente com as mesmas condições de trabalho dos programadores, perfazendo uma infraestrutura de DW já conhecida por todos.

As ferramentas utilizadas foram as mesmas do Experimento I. O SQL Server 2008 e a FGCod, com a diferença de que foi utilizado o módulo de manutenção, desta última.

AMBIENTE CRIADO E ARTEFATOS PRODUZIDOS

Foi definido e criado um ambiente de *Data Warehouse* com os esquemas dimensionais e áreas de *Staging* para cada programador.

5.2 Operação do Experimento

5.2.1 Preparação

A seguir, são enumeradas as etapas de preparação para a execução do experimento.

1) Alocação de programadores a Casos de Uso: foi apresentado, a cada programador, um documento impresso, contendo uma descrição detalhada dos Casos de Uso que seriam utilizados por eles, em caso de eventuais dúvidas;

2) *Revisão de conceitos básicos sobre rotinas de povoamento para o grupo de programadores*: nessa fase, foi realizada uma revisão sobre as rotinas de povoamento para ambientes de DW com os programadores selecionados;

3) *Treinamento da ferramenta FGCod – módulo manutenção*: pela facilidade de uso, foi realizado um treinamento de 30 minutos com os programadores, ministrado por uma pessoa não envolvida com o experimento, a fim de que eles pudessem se familiarizar com a ferramenta de manutenção automática de código.

Em resumo, todos os computadores foram preparados, com as mesmas configurações, para que os programadores estivessem sobre as mesmas condições de trabalho. Além disso, foi apresentado, a cada programador, um documento impresso, contendo uma descrição detalhada dos Casos de Uso que seriam utilizados por eles, em caso de eventuais dúvidas.

5.2.2 Execução

Ao final das etapas anteriores, deu-se início ao experimento, com a realização do sorteio dos dois grupos e posterior execução, seguindo o que foi definido no planejamento.

A avaliação da ferramenta, ao final do experimento, por parte dos profissionais, foi positiva, visto que os mesmos comentaram que a usabilidade da ferramenta contribui para a redução do tempo na manutenção dos procedimentos.

5.2.2.1 Como os dados foram coletados

Foi calculado o tempo gasto por cada programador, para manutenção manual e para manutenção automática, de cada Caso de Uso, levando em consideração o tempo para manutenção e execução. Ao término de todo o experimento, foram armazenados os procedimentos manuais e automáticos para análise de erros.

Os procedimentos mantidos pelos programadores foram avaliados individualmente, comparando-os a um procedimento padrão, considerado como um "modelo" a ser seguido, o qual já possui uma estrutura bem testada e aprovada por empresas. Para execução de cada procedimento, foi analisada a qualidade dos dados carregados para o DW, por um profissional conhecedor do negócio, identificando possíveis inconsistências existentes. Após a detecção de inconsistências, foram atribuídos pesos às mesmas, a depender do grau de criticidade apresentado.

Os resultados desses dados coletados serão apresentados na seção 5.3.

5.2.3 Validação dos Dados

Foram computadas as médias do tempo de manutenção e médias de grau de criticidade de erros encontrados nos dados povoados no DW, após execução dos procedimentos.

Como auxilio para análise, interpretação e validação, foram utilizados três tipos de testes estatísticos, Teste Shapiro-Wilk, Teste T e Teste de Wilcoxon. O Teste Shapiro-Wilk foi utilizado para verificar normalidade das amostras. O Teste T foi utilizado para comparar a média de duas amostras pareadas (WOHLIN et al., 2000) e, por fim, o Teste de Wilcoxon para comparar as médias das amostras pareadas que não obtiveram normalidade nos dados, verificando a magnitude da diferença (WOHLIN et al., 2000).

Todos os testes estatísticos foram feitos utilizando a Ferramenta SPSS – IBM (SPSS, 2014).

5.3 Resultados

5.3.1 Análise e Interpretação de Dados

Para responder às questões, foram analisadas as seguintes variáveis dependentes: a) o tempo para manutenção de cada procedimento e; b) os erros encontrados no povoamento.

Tempo de Criação dos Procedimentos

Para responder à Questão de Pesquisa 01, os resultados relacionados ao tempo de manutenção dos procedimentos por cada participante são apresentados na Tabela 8.

Tabela 8: Tempos de manutenções individuais por programador.

	UC01		UC02	
	Manual (minutos)	**Automática (minutos)**	**Manual (minutos)**	**Automática (minutos)**
Programador1	82	13	12	6
Programador 2	59	17	6	5
Programador 3	88	21	10	9
Programador 4	34	5	22	3
Programador 5	16	3	9	3
Programador 6	22	6	10	8
Programador 7	66	8	13	4
Programador 8	52	10	12	5

Para o Caso de Uso 01 (UC01), os resultados mostram que a média de tempo dos programadores para realizarem manutenção no procedimento de forma manual foi de 52,38 minutos e de forma automática 10,38 minutos. No Caso de Uso 02 (UC02), para manutenção manual, os programadores obtiveram uma média de 11,5 minutos e, de forma automática, 5,38 minutos.

Esses resultados sugerem que a manutenção de procedimentos de forma automática possui, em média, menor tempo de implementação, quando comparada com a manutenção dos mesmos procedimentos de forma manual por programadores com experiência na área. Com isso, a partir desta análise prévia dos dados, supõe-se que a resposta para a Questão 01 de Pesquisa seria “sim”, que a manutenção automática de código pode reduzir o tempo dos programadores no processo de povoamento de um DW, já que a manutenção feita de forma automática obteve uma diferença de tempo relevante, para ambos os Casos de Uso. Todavia, não é possível fazer tal afirmação sem evidências estatísticas suficientemente conclusivas.

Para isto, primeiramente, definimos um nível de significância de 0.05 em todo o experimento e foi aplicado o Teste de Shapiro-Wilk, para análise da distribuição normal das amostras (Tabela 9). O valor da variável Sig., leia-se *p-value*, para manutenção manual, foi 0.683, e 0.657, na manutenção automática. Estes valores são maiores que o nível de

significância adotado. Assume-se que a distribuição dos dados, para ambas as implementações, é normal.

Tabela 9: Teste de Shapiro-Wilk em relação ao tempo de implementação manual para o Caso de Uso 01. (Fonte: Ferramenta SPSS - IBM).

	Shapiro-Wilk		
	Statistic	df	Sig.
UC01ManTempManual	,947	8	,683
UC01ManTempAutomatico	,945	8	,657

Sendo assim, o Teste de hipótese aplicado neste contexto será o Teste T, caracterizado como paramétrico, para amostras emparelhadas. Para isso, verificou-se um Sig. de 0.001, (Tabela 10), fortemente menor que o nível de significância adotado. Desta forma, confirmou-se a evidência de diferença entre as médias de 42,0 (Tabela 10), para o UC01.

Tabela 10: Teste T em relação ao tempo de manutenção – UC01. (Fonte: Ferramenta SPSS - IBM).

		Paired Differences					t	df	**Sig.**
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 1	UC01ManuTempoManual UC01ManuTempoAuto	**42,0**	21,699	7,671	23,858	60,141	5,47	7	**,001**

Posteriormente, foi aplicado o Teste de Shapiro-Wilk, para análise da distribuição normal das amostras, no UC02. O valor da variável Sig., para manutenção manual, foi 0.091, e 0.378, na manutenção automática (Tabela 11). Estes valores são maiores que o nível de significância adotado. Assume-se que a distribuição dos dados, para ambas as implementações, é normal.

Tabela 11: Teste de Shapiro-Wilk em relação ao tempo de implementação manual para o Caso de Uso 01. (Fonte: Ferramenta SPSS - IBM).

	Shapiro-Wilk		
	Statistic	Df	Sig.
UC02ManuTempoManual	,848	8	,091
UC02ManuTempoAuto	,913	8	,378

Desta forma, no UC02, também foi utilizado o Teste T, para o Teste de hipótese com amostras emparelhadas. Para isso, verificou-se um Sig. de 0.018 (Tabela 12), menor que o nível de significância de 0.05. Com isso, confirmou-se a evidência de diferença entre as médias, para o UC02.

Tabela 12: Teste T em relação ao tempo de manutenção – UC02. (Fonte: Ferramenta SPSS - IBM).

		Paired Differences					T	df	**Sig.**
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 1	UC02ManuTempoManual UC02ManuTempoAuto	**6,375**	5,902	2,086	1,440	11,309	3,05	7	**,018**

Diante do apresentado, confirmou-se a evidência de diferença entre as médias, para o UC01 e UC02. O tempo médio de manutenção é significativamente diferente, ou seja, a hipótese (H_0), de que a manutenção automática e manual têm a mesma eficiência foi rejeitada, pois o teste de significância é < 0.05 para ambos Casos de Uso. Consequentemente, não é rejeitada a hipótese alternativa de que a manutenção automática é mais eficiente que a manutenção manual.

Erros de Povoamento dos Dados

Para responder à Questão de Pesquisa 02, foram comparados os erros, contabilizados levando em conta o esforço através da matriz GUT (Gravidade, Urgência e Tendência) (MARSHALL et al., 2011) (Tabela 13), encontrados após execução dos Procedimentos SQL, para os dois Casos de Uso diferentes. Os resultados relacionados ao grau de criticidade de erros dos procedimentos por cada participante são apresentados na Tabela 14. Estes erros foram encontrados com abordagem *blind*, com o auxílio de um usuário de negócio com conhecimento do sistema OLTP, o qual não estava a par do experimento.

Na matriz GUT, Tabela 13, são definidos os problemas que foram encontrados nos procedimentos gerados, a partir da manutenção. Além disso, é estabelecido o valor para os índices de gravidade, urgência e tendência de erros para povoamento de dados em um *DW*.

Tabela 13: Matriz GUT – Experimento II

	Problemas	G	U	T	Grau Crítico
1	Código desnecessário no Script	2	1	2	4
2	Erro na atualização do atributo que representa a data final para uma dimensão com atributo tipo 2.	3	2	2	12
3	Não atualização de dados.	5	5	5	125
4	Duplicação de dados.	4	4	4	64

O primeiro problema detectado, Tabela 13, foi a presença de código no script sem estar sendo utilizado, como, por exemplo, a declaração de uma variável sem futura utilização. Isso acarreta na poluição do código, dificultando ainda mais o entendimento e manutenção do mesmo.

No segundo caso, linha 2 da Tabela 13, é identificado o erro para o tratamento de histórico do atributo de data final, com comportamento tipo 2 (armazenamento de histórico). Esse erro infringe a regra de atualização e armazenamento correto da data final para os registros que não são mais correntes ou válidos atualmente.

O pior caso envolve o problema 3. A não atualização de dados, com o tratamento correto de histórico na dimensão, é uma falta grave e deve ser diagnosticada o quanto antes no processo de povoamento. Por fim, a duplicação de dados, problema 4, está atrelado à inconsistência e redundância em um ambiente de DW. Esta redundância leva a altos custos de armazenamento e acesso aos dados.

Para o Caso de Uso 01 (UC01), os resultados mostram que a média de grau de criticidade de erros dos programadores, para implementarem o procedimento de forma manual, foi 57,75, e, de forma automática, 0 (zero), ou seja, ao serem executados os procedimentos mantidos por cada programador através da ferramenta FGCod, não foram encontrados erros. No Caso de Uso 02 (UC02), para manutenção manual, foi obtida uma média de grau de criticidade de erros de 40,75. Para manutenção automática, a média também foi 0 (zero).

Tabela 14: Grau de criticidade de erros contabilizados para cada programador.

	UC01 - Clientes			UC02 - Produtos		
		Total de grau crítico			Total de grau crítico	
	Problemas	Manual	Automático	Problemas	Manual	Automático
Programador 1	1	4	0	3	4	0
Programador 2	2;3	137	0	1;3	129	0
Programador 3	4	64	0	1	4	0
Programador 4	-	0	0	-	0	0
Programador 5	4	64	0	-	0	0
Programador 6	1	4	0	-	0	0
Programador 7	3	125	0	3	125	0
Programador 8	4	64	0	4	64	0

Pela constância da ausência de erros na manutenção automática e consequente não normalidade dos dados, já que as diferenças destes erros não se distribuem normalmente, foi aplicado o Teste de Wilcoxon como Teste de Hipótese. Esse se caracteriza como não paramétrico, para amostras emparelhadas, levando em consideração que a amostra tem um comportamento contínuo e simétrico. O teste, além de comparar a diferença das amostras, também verifica a magnitude dessa diferença.

Com aplicação do Teste de Wilcoxon, para o UC01, verificou-se que o Sig. de 0.017 (Tabela 15) é menor que o nível de significância de 0.05, ou seja, confirmou-se a evidência de diferença entre as médias de grau de criticidade de erros, para o UC01.

Tabela 15: Teste Wilcoxon em relação aos erros para UC01 – Manutenção (Fonte: Ferramenta SPSS - IBM).

	UC01ErrosManutençãoAuto-UC01ErrosManutençãoManual
Z	-2,388
Asymp. Sig.	**,017**

Para o UC02, também foi aplicado o Teste de Wilcoxon. Verificou-se que o valor de Sig. é 0.042 (Tabela 16), menor que o nível de significância adotado. Assim, confirmou-se a evidência de diferença entre as médias de grau de criticidade de erros, para o UC02.

Tabela 16: Teste Wilcoxon em relação aos erros para UC02 – Manutenção (Fonte: Ferramenta SPSS - IBM).

	UC02ErrosManutençãoAuto-UC02ErrosManutençãoManual
Z	-2,032
Asymp. Sig.	**,042**

Diante dos resultados, confirmam-se evidências de que os erros de implementação de forma manual e automática são significativamente diferentes. Ou seja, a hipótese (H_0), de que a manutenção automática e manual têm a mesma eficácia foi rejeitada, pois o teste de significância é < 0.05 para ambos Casos de Uso. Consequentemente, não foi rejeitada a hipótese alternativa de que a manutenção automática é mais eficaz que a manutenção manual.

5.3.2 Ameaças à validade

Este experimento possui as mesmas ameaças e mitigações que o Experimento I, descrito no tópico 4.3.2 Ameaças à validade.

CAPÍTULO 6

6. Trabalhos Relacionados

Existem, atualmente, muitas ferramentas de ETL *open source* que usam um interpretador para procedimentos em XML. O Pentaho Data Integration (Kettle) (PENTAHO, 2014), por exemplo, é uma ferramenta gráfica a qual salva seus procedimentos em XML. Alguns de seus componentes são exclusivos para cada tipo de banco de dados, dificultando alguns tratamentos de histórico. O Talend (TALEND, 2014) é outra ferramenta muito utilizada, a qual gera código Java ou Perl. O CloverETL Data Integration (CLOVERETL, 2014) também tem seus recursos baseados e armazenados em XML. Além das ferramentas citadas anteriormente, existem várias outras que também utilizam o XML, como por exemplo, DataCleaner (DATACLEANER, 2014), RedHat (REDHAT, 2014) e Apatar (APATAR, 2015).

Algumas ferramentas proprietárias também utilizam XML, é o exemplo da IBM Information Server (Data Stage) (IBM, 2014) e do Oracle Warehouse Builder (OWB) (ORACLE, 2007). Ambas usam XML com suporte OLAP (*On-line Analytical Processing*) para criar o *workspace* analítico e os metadados necessários nos catálogos de banco de dados.

Algumas ferramentas Open Source e proprietárias foram desenvolvidas para auxiliar algumas fases do processo ETL, como as listadas acima. No entanto, como já mencionado anteriormente, muitas empresas optam por codificar várias etapas do processo ETL manualmente, dentre vários outros motivos, com o objetivo de capturar os tratamentos específicos que devem ser dados aos artefatos e usufruir de procedimentos executando dentro do banco.

Diferente das abordagens anteriores, nossa ferramenta gera efetivamente código em uma extensão da linguagem SQL. Desta forma, não foram encontrados, até o momento, trabalhos fortemente relacionados. Em (MUÑOZ; MAZÓN; TRUJILLO, 2009), é apresentada uma abordagem dirigida a modelos para a geração automática de processos ETL. Essa abordagem difere da nossa, uma vez que o objetivo é gerar processos baseados em

modelos de arquitetura de ferramentas ETL já existentes. Nesse sentido, o tratamento dado a dimensões é limitado às ferramentas que se integram ao *framework*.

Em (THOMSEN; PEDERSEN, 2009), é apresentado um *framework* para programação de rotinas ETL. Para avaliação, os participantes do estudo foram os próprios autores. A falta de definição de metadados também é uma fraqueza em relação à nossa abordagem. Os metadados que definem a extração e povoamento são utilizados diretamente no código, dificultando a reutilização.

O código gerado pela FGCod oferece maior flexibilidade para tratar casos específicos de transformações durante as rotinas de povoamento. Este código pode ser facilmente modificado por um programador que conheça uma extensão da linguagem SQL, como T-SQL, PL/SQL, pgSql, etc. Além disso, conforme larga experiência do colaborador escolhido na indústria (12 anos fornecendo soluções BI) e (RANJIT SINGH; KAWALJEET SINGH, 2010), muitas empresas codificam manualmente suas rotinas de povoamento, gerando problemas de qualidade pela falta de uma padronização nessa codificação. Muitos erros foram detectados no experimento, como, por exemplo: eliminação e inserção de registros já encontrados na dimensão; falta de atualização para alguns atributos; não atualização de dados; e, duplicação de dados.

Em (PEREIRA; AGUIAR; SANTOS, 2011) é apresentado uma ferramenta, denominada Gestor de CRUD (CRUD-H), que fornece geração automática de código CRUD com complementar suporte para teste de software e manutenção nas camadas de acesso a dados.

Tem como requisitos não funcionais: desempenho, usabilidade e produtividade. Diz ter uma boa performance, capaz de fornecer mecanismos para codificar criar, ler, atualizar e excluir (CRUD) expressões dentro Strings, incorporando facilmente o poder e a expressividade do SQL. Ela possui um módulo “CRUD-DO Generator” – responsável por criar os códigos automaticamente para as Expressões de CRUD’s desejados. E, além de outros, tem também o módulo “CRUD Maintenance”: responsável por manter todos os CRUD’s existentes, quando precisar editar ou excluir, à medida que for alterado será testado novamente.

Este trabalho se difere do projeto em questão, pelo fato de não estar relacionado com processo de cargas em ambientes BI, no entanto tem pontos em comum, quando se trata da

preocupação de geração de código automático e manutenção, envolvendo SQL, querendo atingir alguns objetivos, como produtividade, desempenho e confiabilidade.

CAPÍTULO 7

7. CONCLUSÕES

7.1 Conclusões

Ambientes de Inteligência Aplicada aos Negócios (*Business Intelligence)* exigem dados organizacionais válidos, consistentes e íntegros. Esses itens de qualidade representam preocupações constantes para as empresas no processo de utilização dos sistemas de suporte à decisão.

Este trabalho apresentou importantes contribuições para redução do tempo de desenvolvimento e qualidade na ES para carga de DWs, bem como fomenta a experimentação em ambiente industrial. A ferramenta passou a ser usada pelas empresas atendidas pelo parceiro escolhido, considerando a experimentação e análise dos resultados realizadas, bem como perfazendo uma inovação para quem adota esta abordagem.

Vale ressaltar que a execução segura e eficiente de procedimentos em SQL diretamente no Banco de Dados é uma opção considerada por grande parte da indústria, necessitando de ferramentas de apoio a este tipo de abordagem em Engenharia de Software.

Há diversos benefícios em utilizar *Stored Procedures* neste contexto, pois além de melhorar a performance e criar mecanismos de segurança entre a manipulação dos dados, pode reduzir o tráfego na rede, visto que os comandos são executados diretamente no servidor, que é isolado e dedicado somente para execução dos procedimentos. Existem críticas, em sites sem embasamento científico, quanto ao uso de *stored procedures* para cargas, contudo, as críticas não se aplicam ao estado da arte em SGBDs e ao contexto de cargas para um DW, as quais acontecem utilizando dados já presentes no banco, o que favorece a execução de procedimentos diretamente no SGBD.

Dado este contexto e a inovação da abordagem, a apresentação do experimento I e II embasará a adoção da mesma ou a criação de uma abordagem similar por empresas que utilizam este tipo de estratégia. Na análise dos resultados, apesar de terem sido usados

programadores acostumados a escrever processos de carga, houve evidencias da redução do tempo e eliminação dos erros de codificação durante a fase de construção e manutenção de ETL mais complexos para ambientes de BI. Isto denota um benefício ainda maior para programadores iniciantes em ETL.

7.1.1 Contribuições

As contribuições obtidas são:

- Abordagem para automatizar a criação e manutenção de cargas de Ambientes BI alimentados por DWs;
- *Framework* para satisfazer os diferentes dialetos SQL;
- Uso da ferramenta por parceiros da empresa que colaborou com o primeiro experimento, propiciando uma melhor eficiência e eficácia na criação e manutenção dos códigos SQL;
- Escrita de Artigo, o qual foi submetido e aceito em Congresso.

7.2 Trabalhos Futuros

Alguns pontos deste projeto são importantes e ainda precisam ser explorados. Primeiramente, como trabalhos futuros, pretende-se estender a abordagem para diversos idiomas SQL, já que os experimentos realizados até o momento foram somente para o T-SQL. Ato contínuo, serão feitas análises comparando o padrão do código SQL gerado pela ferramenta e o código feito aleatoriamente pelos programadores, averiguando se o uso deste padrão pode trazer algum benefício para as Empresas.

8. REFERÊNCIAS

ABRANTES, J. **Gestão da Qualidade**. 1ª. ed. Rio de Janeiro: Interciência, 2009.

APATAR. **APATAR Connecting data**, 2015. Disponível em: <http://www.apatar.com/>

BASILI, V. R.; WEISS, D. M. A Methodology for Collecting Valid Software Engineering Data. **IEEE Transactions on Software Engineering**, v. SE-10, n. 6, p. 728–738, 1984.

CHAUDHURI, S.; DAYAL, U.; NARASAYYA, V. An overview of business intelligence technology. **Communications of the ACM**, v. 54, n. 8, p. 88, 2011.

CHEN, H.; STOREY, V. C. B USINESS I NTELLIGENCE AND A NALYTICS : F ROM B IG D ATA TO B IG I MPACT. v. 36, n. 4, p. 1165–1188, 2012.

CLOVERETL. **Data Integration Software**, 2014. Disponível em: <www.cloveretl.com>

COLAÇO, M. J. **Projetando sistemas de apoio à decisão baseados em data warehouse**. 1. ed. Rio de Janeiro: Books, Axcel, 2004.

DATACLEANER. **The premier data quality solution**, 2014. Disponível em: <www.datacleaner.org>

DAVENPORT, T. H. Competing on Analytics Competing on Analytics. **Harvard Business Review**, v. 84, n. 1, p. 98–107, 2006.

DOAN, A.; RAMAKRISHNAN, R.; HALEVY, A. Y. Crowdsourcing systems on the World-Wide Web. **Communications of the ACM**, v. 54, n. 4, p. 86, 2011.

GONÇALVES, L. F. V. A REDUÇÃO DE PROBLEMAS DE UTILIZAÇÃO DO MÉTODO CICLO PDCA : UM ESTUDO DE CASO NA. **VII Congresso Nacional de Escelência em Gestão**, p. 18, 2011.

IBM. **IBM, Information Server (Data Stage)**, 2014. Disponível em: <http://goo.gl/H4j8v1>

IDRIS, N.; AHMAD, K. Managing Data Source quality for data warehouse in manufacturing services. **Proceedings of the 2011 International Conference on Electrical Engineering and Informatics**, n. July, p. 1–6, 2011.

IEEE. INTERNATIONAL STANDARD ISO / IEC / IEEE. v. 2010, p. 418, 2010.

IMHOFF CLAUDIA, GALEMMO NICHOLAS, G. G. J. **Mastering Data Warehouse Design: Relational and Dimensional Techniques**. Indianapolis: Wiley Publish, 2003.

INMON, W. H. **Building the Data Warehouse**. 4. ed. Indianapolis, Indiana: Wiley Publishing Inc., 2005.

JURISTO, N.; MORENO, A. M. Basics of Software Engineering Experimentation. **Analysis**, v. 5/6, p. 420, 2001.

KERLINGER, F. N.; LEE, H. B. **Foundations of Behavioral Research**. 2ª. ed. [s.l.] Cengage Learning, 1973.

KIMBALL, R. et al. The data warehouse lifecycle toolkit: practical techniques for building data warehouse and business intelligence systems. p. 636 p., 2008.

KIMBALL, R.; ROSS, M. **The data warehouse toolkit: The complete Guide to Dimensional Modeling**. 2. ed. [s.l.] John Wiley and Sons, Inc., 2002.

MARSHALL, I. J. et al. **Gestão da qualidade**. 10. ed. Rio de Janeiro: FGV, 2011.

MAZANEC, M.; MACEK, O. On General-purpose Textual Modeling Languages On General-purpose Textual Modeling Languages. p. 1–12, 2012.

MICROSOFT. **Centro de Desenvolvedor Microsoft**, 2015. Disponível em: <https://msdn.microsoft.com/developer-centers-msdn>

MUÑOZ, L.; MAZÓN, J.-N.; TRUJILLO, J. Automatic generation of ETL processes from conceptual models. **Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP**, p. 8, 2009.

ORACLE. **Oracle Warehouse Builder**, 2007. Disponível em: <http://goo.gl/Md4hjn>

ORNE, M. T. Demand Characteristics and the Concept of Quasi-Controls. In: **Artifacts in Behavioral Research**. [s.l.] Oxford Scholarship, 2009. p. 912.

OSBORNE, W. M.; CHIKOFDRY, E. J. F itting Pieces to the M a intenance Puzzle. **IEEE Software**, v. 7, p. 11–12, 1990.

PENTAHO. **PENTAHO Open Source Business Intelligence**, 2014. Disponível em: <http://community.pentaho.com/projects/data-integration/>

PEREIRA, O. M.; AGUIAR, R. L.; SANTOS, M. Y. CRUD-DOM: A model for bridging the gap between the object-oriented and the relational paradigms. **Proceedings - 5th International Conference on Software Engineering Advances, ICSEA 2010**, v. 4, n. 1, p. 114–122, 2011.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 2007.

RANJIT SINGH; KAWALJEET SINGH. A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing. **International Journal of Computer Science Issues**, v. 7, n. 3, No 2, p. 41–50, 2010.

REDHAT. **REDHAT Application Development and Integration**, 2014. Disponível em: <http://www.redhat.com>

RIBEIRO, L. DE S.; GOLDSCHMIDT, R. R.; CAVALCANTI, M. C. Complementing data in the ETL process. **DaWaK'11 Proceedings of the 13th international conference on Data warehousing and knowledge discovery**, p. 112–123, 2011.

SANTOS, V.; BELO, O. Slowly Changing Dimensions Specification a Relational Algebra Approach. **International Journal on Information …**, v. 01, n. 03, 2011.

SEVERINO, A. J. **Metodologia do trabalho científico**. 23. ed. [s.l.] Editora, Cortez, 2008.

SMITH, H.; FINGAR, P. **Business process management: the third wave**. 1. ed. [s.l.] Meghan-Kiffer Press, 2003.

SOMMERVILLE, I. **Software Engineering**. England: Pearson Education Limited, 2001.

SOMMERVILLE, I. **Software Engineering**. 8. ed. São Paulo: Pearson, 2006.

SPSS. **IBM SPSS Statistics**, 2014. Disponível em: <http://goo.gl/eXfcT3>

SQLSERVER. **SQL SERVER Management Studio**, 2008. Disponível em: <www.microsoft.com/SQLServer>

TALEND. **TALEND Open Integration Solutions**, 2014. Disponível em: <www.talend.com>

TAYI, G.; BALLOW, D. Communications of the Examining data. **ACM**, v. 41, 1998.

THOMSEN, C.; PEDERSEN, T. B. pygrametl : A Powerful Programming Framework for Extract – Transform – Load Programmers. 2009.

TRISTÃO, R. G. C. **A importância das ações corretivas e ações preventivas nos sistemas de gestão da qualidade - um estudo em empresas certificadas ISO 9001 no estado do rio de janeiro**, 2011.

WOHLIN, C. et al. **Experimentation in software engineering: An introduction**. USA: Kluwer Academic Publishers, 2000. v. 40

9. APÊNDICES

9.1 Apêndice I – Publicações Produzidas

Artigos Publicados

- Desenvolvimento e Avaliação de uma Ferramenta de Geração Automática de Código para Ambientes de Apoio à Decisão. ERBASE, 2012.
- Geração Automática de Código SQL para Dimensões em um Ambiente de *Data Warehouse.* 2ª Semana de Informática (SI), 2012.
- Experimentação na Indústria para Aumento da Efetividade da Construção de Procedimentos ETL em um Ambiente de *Business Intelligence*. Simpósio Brasileiro de Sistemas de Informação (SBSI), 2015.

Palestra

- Experimentação na Indústria para Aumento da Efetividade da Construção de Procedimentos ETL em um Ambiente de *Business Intelligence.* Disciplina de Seminários –UFS, 2014.

9.2 Apêndice II – Instruções passadas para programadores no Experimento I

1 - Caso de Uso 1 – Clientes

Objetivo: Gerar Procedimento para realizar cargas da Área de *Staging* (**staging.TB_Aux_Cliente**) para a Dimensão (**dw.DIM_Cliente**).

Para este caso, a dimensão não terá armazenamento de histórico. Ou seja, todos os atributos terão comportamento do **Tipo 1**.

Ao ser feita uma carga os valores **sobrescrevem o valor antigo** de uma linha da dimensão pelo valor atual proveniente da Área de *Staging*. Dessa forma, o atributo sempre irá refletir o valor mais recente proveniente do ambiente operacional.

Para atualização do atributo é utilizada a chave natural, uma vez que esta deve permanecer inviolável e constante. Afinal, ela servirá como referência entre os ambientes transacional e de suporte à decisão para a atualização dos dados. Ao serem identificados novos valores, o procedimento cria um novo registro no DW.

Segue, figura 1, um fluxograma representando o fluxo do procedimento de povoamento para atributos do tipo 1.

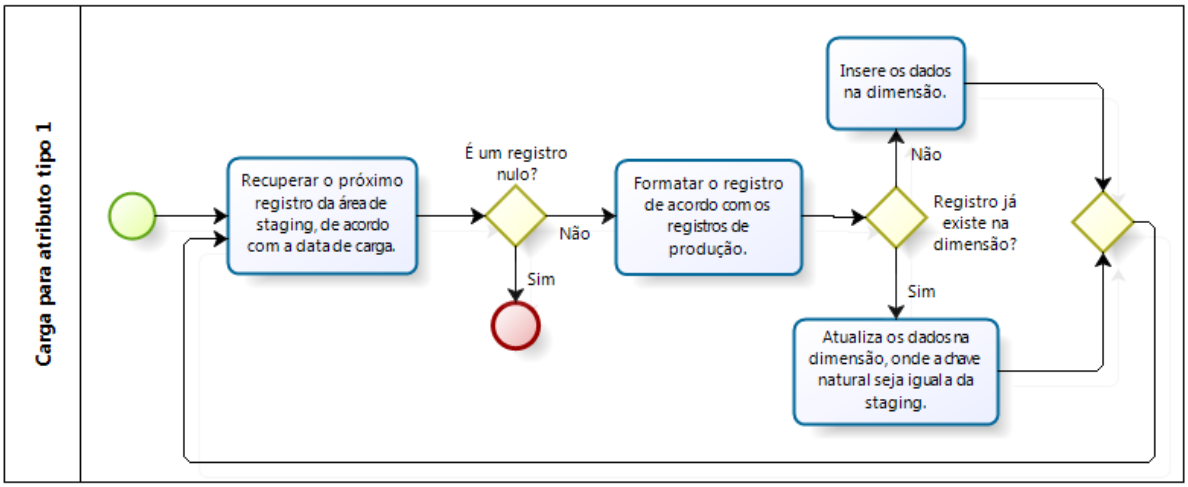

Figura 1 - Fluxograma do algoritmo de povoamento para atributos do tipo 1 (Igor Peterson).

A tabela 1 caracteriza a dimensão e os atributos quanto ao **armazenamento de histórico.**

Nome da dimensão	dw.DIM_Cliente	
Tipo de tratamento de histórico	tipo 1	
Atributo	**Tipo**	**Papel do Atributo**
id_cliente		Data de Carregamento
cod_cliente		Chave natural
Cpf	1	
nome_cliente	1	
data_nascimento	1	
endereco	1	
Bairro	1	
Cidade	1	

Tabela 1 – Característica da dimensão dw.DIM_Cliente e seus atributos quanto ao armazenamento de histórico.

A tabela 2 ilustra a **especificação** dos atributos da tabela da Área de *Staging* staging.TB_Aux_Cliente.

staging.TB_Aux_Cliente						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
`data_carga`	*Datetime*				*Not null*	Data de Carregamento
`cod_cliente`	*Int*				*Not null*	Chave natural
`cpf`	*Varchar*	11			*Not null*	
`nome_cliente`	*varchar*	50			*Not null*	
`data_nascimento`	*datetime*				*Not null*	
`endereco`	*varchar*	50			*Not null*	
`bairro`	*varchar*	20			*Not null*	
`cidade`	*varchar*	20			*Not null*	

Tabela 2 – Especificação dos atributos da Tabela staging.TB_Aux_Cliente.

A tabela 3 ilustra a **especificação** dos atributos da tabela Dimensão dw.DIM_Cliente.

dw.DIM_Cliente						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
id_cliente	*int*				*Not null*	Surrogate Key
cod_cliente	int				*Not null*	Chave Natural
cpf	*varchar*	11			*Not null*	
nome_cliente	*varchar*	50			*Not null*	
data_nascimento	datetime				*Not null*	
endereco	*varchar*	50			*Not null*	
bairro	*varchar*	20			*Not null*	
cidade	*varchar*	20			*Not null*	

Tabela 3 – Especificação dos atributos da Tabela dw.DIM_Cliente.

A Tabela 4 ilustra a **relação dos atributos** da área de *Staging* (staging.TB_Aux_Cliente) com os atributos da Dimensão (dw.DIM_Cliente). Ou seja, qual atributo da Tabela Auxiliar de Cliente corresponde ao atributo da Dimensão Cliente.

Cliente	
Atributo da staging.TB_Aux_Cliente	**Atributo da dw.DIM_Cliente**
cod_cliente	cod_cliente
cpf	cpf
nome_cliente	nome_cliente
data_nascimento	data_nascimento
endereco	endereco
Bairro	bairro
Cidade	cidade

Tabela 4 – Relação dos atributos da Área de *Staging* com os atributos da Dimensão.

OBSERVAÇÃO:

Executar o seu procedimento (manual e automático) para as seguintes datas, nesta ordem:

'20131105'

'20131110'

2 Caso de Uso 2 – Produtos

Objetivo: Gerar Procedimento para realizar cargas da Área de *Staging* (**staging.TB_Aux_Produto**) para a Dimensão (**dw.DIM_Produto**).

Para este caso, a dimensão **terá atributos com armazenamento de histórico tipo 2** para alguns atributos. Alguns atributos da dimensão não armazenarão histórico, ou seja, serão do tipo 1, outros atributos armazenarão histórico de acordo com os critérios apresentados no armazenamento de histórico do tipo 2 (A tabela 5 ilustra qual o tipo de armazenamento de histórico de cada atributo).

O tratamento de histórico do tipo 2 é realizado através da **criação de novos registros**. É gerado um novo valor do identificador da dimensão (*Surrogate Key*) quando há alteração em atributos que necessitam armazenar todo o seu histórico.

No esquema da dimensão que possui atributos do Tipo 2 **surgem novos atributos** que não existem na dimensão que contempla somente atributos do Tipo 1. Esses novos atributos são utilizados para o tratamento dado ao histórico dos atributos do tipo 2. São eles: a data inicial, que representa a data em que o registro foi gravado; a data final, que representa a data em que o registro deixou de ser corrente; e por fim, o atributo que identifica se é um registro atual ou não. Estes são apresentados respectivamente na tabela 07 com os nomes de: **dt_inicio; dt_fim e fl_corrente**.

Quando um registro é inserido, um novo identificador (*Surrogate Key*) é criado e os dados são carregados para a dimensão. Com isso, o atributo dt_inicio armazena a data de quando o registro foi inserido na dimensão. No atributo dt_fim é registrado a ausência de valor (NULL), até que o mesmo deixe de ser atual. E por fim, há o fl_Corrente que apresenta o valor ‘S’ para identificar o registro atual. Quando os atributos que armazenam histórico sofrem alterações, o atributo dt_fim irá registrar a data da alteração. Consequentemente o valor do atributo fl_corrente será alterado para ‘N’, o que indica que aquele registro deixou de ser atual. Uma nova linha, com um novo identificador (*Surrogate Key*), é inserida com as alterações feitas.

Segue, figura 2, um fluxograma representando o fluxo do procedimento de povoamento para atributos do tipo 2.

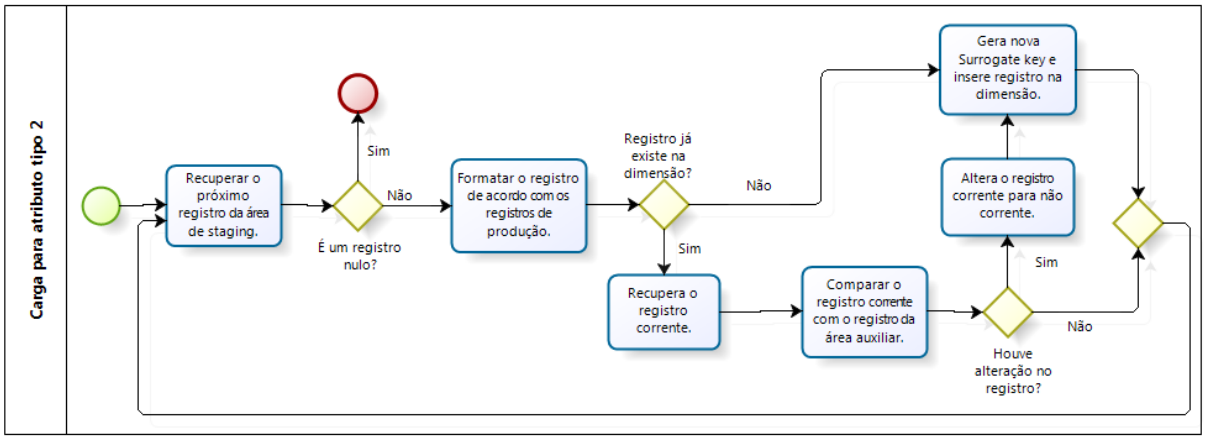

Figura 2 - Fluxograma do algoritmo de povoamento para atributos do tipo 2 (Igor Peterson).

A tabela 5 caracteriza a dimensão e os atributos quanto ao **armazenamento de histórico.**

Nome da dimensão	dw.DIM_Produto	
Tipo de tratamento de histórico	Tipo 1 e 2	
Atributo	**Tipo de Histórico**	**Papel do Atributo**
id_produto		Surrogate Key
codigo_produto		Chave Natural
codigo_barra	1	
descricao	2	
Linha	2	
Valor	2	
dt_inicio		Data Inicial
dt_fim		Data Final
fl_corrente		Flag Corrente

Tabela 5 - Característica da dimensão dw.DIM_Produto e seus atributos quanto ao armazenamento de histórico.

A tabela 2 ilustra a **especificação** dos atributos da tabela da Área de *Staging* staging.TB_Aux_Produto.

staging.TB_Aux_Produto						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
`data_carga`	*datetime*				*Not null*	Data de Carregamento

codigo_produto	*int*				*Not null*	Chave natural
codigo_barra	*int*				*Not null*	
descricao	*varchar*	25			*Not null*	
Linha	*varchar*	20			*Not null*	
Valor	*numeric*		10	2	*Not null*	

Tabela 6 – Especificação dos atributos da Tabela staging.TB_Aux_Produto.

A tabela 7 ilustra a **especificação** dos atributos da tabela Dimensão dw.DIM_Produto.

dw.DIM_Produto						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
id_produto	*int*				*Not null*	Surrogate Key
codigo_produto	int				*Not null*	Chave Natural
codigo_barra	int				*Not null*	
descricao	*varchar*	25			*Not null*	
Linha	*varchar*	20			*Not null*	
Valor	numeric		10	2	*Not null*	
dt_inicio	datetime				*Not null*	Data Inicial
dt_fim	datetime				*Null*	Data Final
fl_corrente	char	1			*Not null*	Flag Corrente

Tabela 7 – Especificação dos atributos da Tabela dw.DIM_Produto.

A Tabela 8 ilustra a **relação dos atributos** da área de *Staging* (staging.TB_Aux_Produto) com os atributos da Dimensão (dw.DIM_Produto). Ou seja, qual atributo da Tabela Auxiliar de Produto corresponde ao atributo da Dimensão Produto.

Produto	
Atributo da staging.TB_Aux_Produto	**Atributo da dw.DIM_Produto**
codigo_produto	codigo_produto
codigo_barra	codigo_barra
descricao	descricao
linha	linha
valor	valor

Tabela 8 – Relação dos atributos da Área de *Staging* com os atributos da Dimensão.

OBSERVAÇÃO:

Executar o seu procedimento (manual e automático) para as seguintes datas, nesta ordem:

'20131105'

'20131110'

9.3 Apêndice III – Instruções passadas para programadores no Experimento II

1 - Caso de Uso 1 - Clientes

Objetivo: Dar Manutenção no Procedimento "CarregaCliente" que realiza cargas da Área de *Staging* (**staging.TB_Aux_Cliente**) para a Dimensão (**dw.DIM_Cliente**).

Executar o seguinte Casos de Uso:

[UC 01] Manter Atributos Cliente	
DESCRIÇÃO:	Alteração e Inclusão de Atributos no procedimento "CarregaCliente".
Ator:	DBA (Administrador de banco de dados) e Programadores
Prioridade:	Essencial
Entradas e Pré-condições:	Ter o Procedimento já criado para carregar dados dos Clientes da área de *Staging* para o DW.
Saídas e pós-condições:	- Atributos referentes à Endereço (endereço, bairro e cidade) alterados o Tipo de histórico, no procedimento. - Atributo referente à Telefone Incluso no procedimento. Atributo referente à Estado Excluído do procedimento.
Fluxo básico:	1. O procedimento "CarregaCliente" necessita de manutenção, para suprir necessidades de Negócio. 2. Será necessário executar os seguintes Subfluxos: e. Subfluxo Alterar f. Subfluxo Incluir g. Subfluxo Excluir
Subfluxo Alterar	1. Ao ser feita carga de dados da área de Staging (tabela auxiliar) para a Dimensão Cliente (**dw.DIM_Cliente)** será necessário armazenar histórico do endereço do Cliente. 2. O Tipo de histórico necessário é o Tipo 2. 3. Os Seguintes atributos devem ser alterados para armazenamento do Tipo 2: - "endereco" - "bairro" - "cidade"
Subfluxo Incluir	1. Uma nova Coluna foi adicionada para Clientes. 2. A inclusão de um Atributo para armazenar telefone, foi incluso na Tabela de **"staging.TB_Aux_Cliente" e "dw.DIM_Cliente".** 3. O atributo referente ao telefone, tanto na **"staging.TB_Aux_Cliente"** como na **"dw.DIM_Cliente"** é "telefone". 4. O Atributo "telefone" não tem armazenamento de histórico, ou seja, é do Tipo 1.
Subfluxo Excluir	5. Uma nova Coluna foi excluída para Clientes. 6. A exclusão do Atributo para armazenar o Estado do Cliente, foi excluído da Tabela de "staging.TB_Aux_Cliente" e "dw.DIM_Cliente". 7. O atributo referente à Estado do Cliente, tanto na

	"staging.TB_Aux_Cliente" como na "dw.DIM_Cliente" era "estado".

OBSERVAÇÃO:

Executar o seu procedimento, após manutenção (manual e automática) para as seguintes datas, nesta ordem:

'20131105'

'20131110'

'20131115'

2- Caso de Uso 2 – Produtos

Objetivo: Dar Manutenção no Procedimento "CarregaProduto", que realiza cargas da Área de *Staging* (**staging.TB_Aux_Produto**) para a Dimensão (**dw.DIM_Produto**).

Executar o seguinte Casos de Uso:

[UC 02] Manter Atributos Produto	
DESCRIÇÃO:	Alteração, Inclusão e Exclusão de Atributos no procedimento "CarregaProduto".
Ator:	DBA (Administrador de banco de dados) e Programadores
Prioridade:	Essencial
Entradas e Pré-condições:	Ter o Procedimento já criado para carregar dados dos Produtos da área de *Staging* para o DW.
Saídas e pós-condições:	- Atributo referente a Linha do Produto alterado o Tipo de histórico. - Atributo referente à Fornecedor Incluso. - Atributo referente à Composição do Produto Excluído.
Fluxo básico:	3. O procedimento "CarregaProduto" necessita de manutenção, para suprir necessidades de Negócio. 4. Será necessário executar os seguintes Subfluxos: c. Subfluxo Alterar d. Subfluxo Incluir h. Subfluxo Excluir
Subfluxo Alterar	1. Ao ser feita carga de dados da área de Staging (**staging.TB_Aux_Produto**) para a Dimensão Produto (**dw.DIM_Produto)** não será mais necessário armazenar histórico para a Linha do Produto. 2. O Tipo de histórico necessário é o Tipo 1. 3. O Seguinte atributo deve ser alterado para armazenamento do

	Tipo 1: - “linha”
Subfluxo Incluir	1. Uma nova Coluna foi adicionada para Produtos. 2. A inclusão de um Atributo para armazenar Fornecedor do Produto, foi incluído na Tabela de **“staging.TB_Aux_ Produto” e “dw.DIM_ Produto”.** 3. O atributo referente ao Fornecedor, tanto na **“staging.TB_Aux_Produto”** como na **“dw.DIM_Produto”** é “fornecedor”. 4. O Atributo “fornecedor” terá armazenamento de histórico, Tipo 2.
Subfluxo Excluir	1. Uma nova Coluna foi excluída para Produtos. 2. A exclusão do Atributo para armazenar a Composição do Produto, foi excluído da Tabela de “staging.TB_Aux_Produto” e “dw.DIM_Produto”. 3. O atributo referente à Composição do Produto, tanto na “staging.TB_Aux_Produto” como na “dw.DIM_Produto” era “composicao”.

OBSERVAÇÃO:
Executar o seu procedimento, após manutenção (manual e automática) para as seguintes datas, nesta ordem:
'20131105'
'20131110'
'20131115'

3 - Descrição do Procedimento “CarregaCliente”

Neste Procedimento, a dimensão **tem armazenamento de histórico tipo 2** para alguns atributos. Alguns atributos da dimensão não armazenarão histórico, ou seja, serão do tipo 1, outros atributos armazenarão histórico de acordo com os critérios apresentados no armazenamento de histórico do tipo 2 (A tabela 1 ilustra qual o tipo de armazenamento de histórico de cada atributo).

A tabela 1 caracteriza a dimensão e os atributos quanto ao **armazenamento de histórico.**

Nome da dimensão	dw.DIM_Cliente	
Tipo de tratamento de histórico	tipo 1 e 2	
Atributo	**Tipo**	**Papel do Atributo**
data_carga		Data de Carregamento
cod_cliente		Chave natural
cpf	1	
nome_cliente	1	
data_nascimento	1	

endereco	2	
bairro	2	
cidade	2	
telefone	1	
dt_inicio		Data Inicial
dt_fim		Data Final
fl_corrente		Flag Corrente

Tabela 1 – Característica da dimensão dw.DIM_Cliente e seus atributos quanto ao armazenamento de histórico.

A tabela 2 ilustra a **especificação** dos atributos da tabela da Área de *Staging* staging.TB_Aux_Cliente.

Staging.TB_Aux_Cliente						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
data_carga	*datetime*				*Not null*	Data de Carregamento
cod_cliente	*int*				*Not null*	Chave natural
cpf	*varchar*	11			*Not null*	
nome_cliente	*varchar*	50			*Not null*	
data_nascimento	*datetime*				*Not null*	
endereco	*varchar*	50			*Not null*	
bairro	*varchar*	20			*Not null*	
cidade	*varchar*	20			*Not null*	
telefone	*varchar*	20			*Not null*	

Tabela 2 – Especificação dos atributos da Tabela staging.TB_Aux_Cliente.

A tabela 3 ilustra a **especificação** dos atributos da tabela Dimensão dw.DIM_Cliente.

dw.DIM_Cliente						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
id_cliente	*int*				*Not null*	Surrogate Key
cod_cliente	int				*Not null*	Chave Natural
cpf	*varchar*	11			*Not null*	
nome_cliente	*varchar*	50			*Not null*	
data_nascimento	datetime				*Not null*	
endereco	*varchar*	50			*Not null*	
bairro	*varchar*	20			*Not null*	
cidade	*varchar*	20			*Not null*	
telefone	*varchar*	20			*Not null*	
dt_inicio	datetime				*Not null*	Data Inicial
dt_fim	datetime				*Null*	Data Final
fl_corrente	char	1			*Not null*	Flag Corrente

Tabela 3 – Especificação dos atributos da Tabela dw.DIM_Cliente.

A Tabela 4 ilustra a **relação dos atributos** da área de *Staging* (staging.TB_Aux_Cliente) com os atributos da Dimensão (dw.DIM_Cliente). Ou seja, qual atributo da Tabela Auxiliar de Cliente corresponde ao atributo da Dimensão Cliente.

Cliente	
Atributo da staging.TB_Aux_Cliente	**Atributo da dw.DIM_Cliente**
cod_cliente	cod_cliente
cpf	cpf
nome_cliente	nome_cliente
data_nascimento	data_nascimento
endereco	endereco
bairro	bairro
cidade	cidade
telefone	telefone

Tabela 4 – Relação dos atributos da Área de *Staging* com os atributos da Dimensão.

4 - Descrição do Procedimento "CarregaProduto"

Para este caso, a dimensão **tem armazenamento de histórico tipo 2** para alguns atributos. Alguns atributos da dimensão não armazenarão histórico, ou seja, serão do tipo 1, outros atributos armazenarão histórico de acordo com os critérios apresentados no armazenamento de histórico do tipo 2 (A tabela 5 ilustra qual o tipo de armazenamento de histórico de cada atributo).

A tabela 5 caracteriza a dimensão e os atributos quanto ao **armazenamento de histórico.**

Nome da dimensão	dw.DIM_Produto	
Tipo de tratamento de histórico	Tipo 1 e 2	
Atributo	**Tipo de Histórico**	**Papel do Atributo**
id_produto		Surrogate Key
codigo_produto		Chave Natural
codigo_barra	1	
descricao	2	
linha	1	
valor	2	
fornecedor	2	
dt_inicio		Data Inicial
dt_fim		Data Final
fl_corrente		Flag Corrente

Tabela 5 – Característica da dimensão dw.DIM_Produto e seus atributos quanto ao armazenamento de histórico.

A tabela 6 ilustra a **especificação** dos atributos da tabela da Área de *Staging* staging.TB_Aux_Produto.

Staging.TB_Aux_Produto						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
data_carga	*datetime*				*Not null*	Data de Carregamento
codigo_produto	*int*				*Not null*	Chave natural
codigo_barra	*int*				*Not null*	
descricao	*varchar*	25			*Not null*	
linha	*varchar*	20			*Not null*	
valor	*numeric*		10	2	*Not null*	
fornecedor	*varchar*	20		2	*Not null*	

Tabela 6 – Especificação dos atributos da Tabela staging.TB_Aux_Produto.

A tabela 7 ilustra a **especificação** dos atributos da tabela Dimensão dw.DIM_Produto.

Dw.DIM_Produto						
Atributo	**Tipo**	**Tamanho**	**Precisão**	**Escala**	**Nulidade**	**Papel do Atributo**
id_produto	*int*				*Not null*	Surrogate Key
codigo_produto	int				*Not null*	Chave Natural
codigo_barra	int				*Not null*	
descricao	*varchar*	25			*Not null*	
linha	*varchar*	20			*Not null*	
valor	numeric		10	2	*Not null*	
fornecedor	*varchar*	20			*Not null*	
dt_inicio	datetime				*Not null*	Data Inicial
dt_fim	datetime				*Null*	Data Final
fl_corrente	char	1			*Not null*	Flag Corrente

Tabela 7 – Especificação dos atributos da Tabela dw.DIM_Produto.

A Tabela 8 ilustra a **relação dos atributos** da área de *Staging* (staging.TB_Aux_Produto) com os atributos da Dimensão (dw.DIM_Produto). Ou seja, qual atributo da Tabela Auxiliar de Produto corresponde ao atributo da Dimensão Produto.

Produto	
Atributo da staging.TB_Aux_Produto	**Atributo da dw.DIM_Produto**
codigo_produto	codigo_produto
codigo_barra	codigo_barra
descricao	descricao
linha	linha
valor	valor
fornecedor	fornecedor

Tabela 8 – Relação dos atributos da Área de *Staging* com os atributos da Dimensão.

5- Como Funciona o armazenamento de Histórico do Tipo 1?

Ao ser feita uma carga os valores **sobrescrevem o valor antigo** de uma linha da dimensão pelo valor atual proveniente da Área de *Staging*. Dessa forma, o atributo sempre irá refletir o valor mais recente proveniente do ambiente operacional.

Para atualização do atributo é utilizada a chave natural, uma vez que esta deve permanecer inviolável e constante. Afinal, ela serve como referência entre os ambientes transacional e de suporte à decisão para a atualização dos dados. Ao serem identificados novos valores, o procedimento cria um novo registro no DW.

Segue, figura 1, um fluxograma representando o fluxo do procedimento de povoamento para atributos do tipo 1.

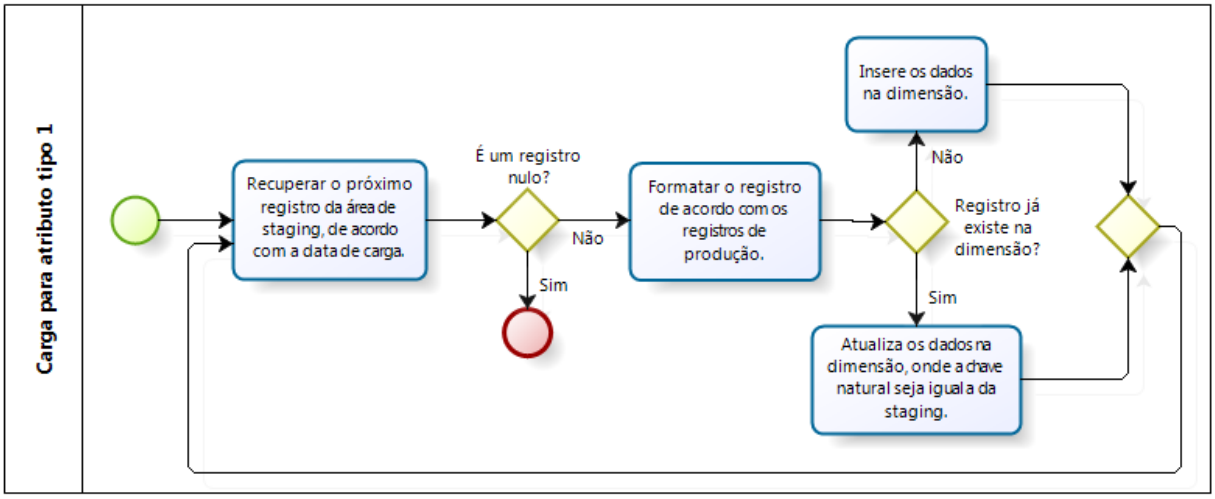

Figura 1 - Fluxograma do algoritmo de povoamento para atributos do tipo 1 (Igor Peterson).

6 - Como Funciona o armazenamento de Histórico do Tipo 2?

O tratamento de histórico do tipo 2 é realizado através da **criação de novos registros**. É gerado um novo valor do identificador da dimensão (*Surrogate Key*) quando há alteração em atributos que necessitam armazenar todo o seu histórico.

No esquema da dimensão que possui atributos do Tipo 2 **surgem novos atributos** que não existem na dimensão que contempla somente atributos do Tipo 1. Esses novos atributos são utilizados para o tratamento dado ao histórico dos atributos do tipo 2. São eles: a data inicial, que representa a data em que o registro foi gravado; a data final, que representa a data em que o registro deixou de ser corrente; e por fim, o atributo que identifica se é um registro

atual ou não. Estes são apresentados respectivamente na tabela 07 com os nomes de: **dt_inicio; dt_fim e fl_corrente**.

Quando um registro é inserido, um novo identificador (*Surrogate Key*) é criado e os dados são carregados para a dimensão. Com isso, o atributo dt_inicio armazena a data de quando o registro foi inserido na dimensão. No atributo dt_fim é registrado a ausência de valor (NULL), até que o mesmo deixe de ser atual. E por fim, há o fl_Corrente que apresenta o valor 'S' para identificar o registro atual. Quando os atributos que armazenam histórico sofrem alterações, o atributo dt_fim irá registrar a data da alteração. Consequentemente o valor do atributo fl_corrente será alterado para 'N', o que indica que aquele registro deixou de ser atual. Uma nova linha, com um novo identificador (*Surrogate Key*), é inserida com as alterações feitas.

Segue, figura 2, um fluxograma representando o fluxo do procedimento de povoamento para atributos do tipo 2.

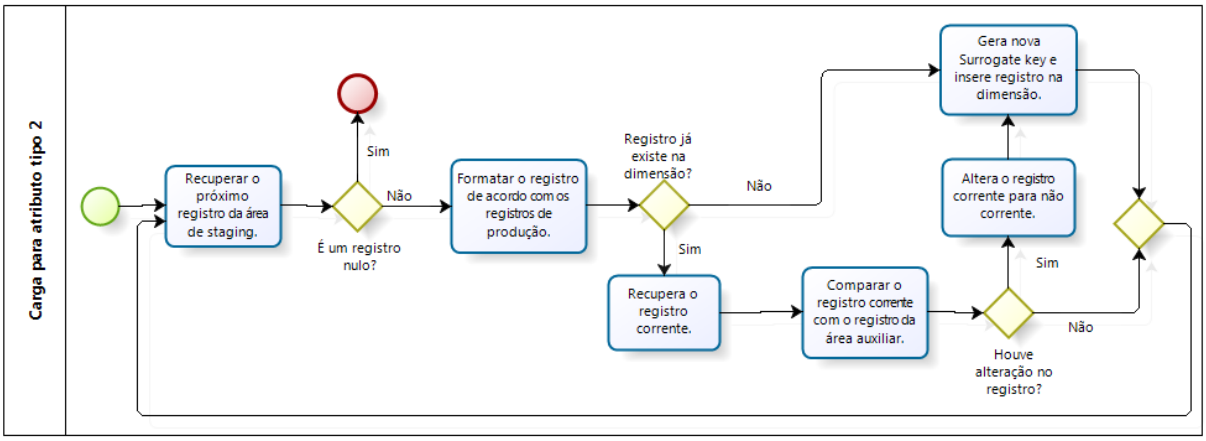

Figura 2 - Fluxograma do algoritmo de povoamento para atributos do tipo 2 (Igor Peterson).

9.4 Apêndice IV – Protocolo para pesquisa Sistemática

Objetivo:

Executar uma revisão sistemática cujo *objetivo* será identificar abordagens para responder perguntas com relação à geração automática de código para cargas de dados em um *Data Warehouse*.

Formulação de Perguntas:

µ0: Quais vantagens apresentadas no uso de *Stored Procedures* em ambiente de carga de DW?

µ1: Quais problemas ainda persistem em *Business intelligence and analytics/Big data*?

µ2: Quais problemas/desvantagens e vantagens/benefícios de carga para DW utilizando XML?

µ3: Quais problemas de qualidade são encontrados nas cargas de um DW?

µ4: Quais os tipos de cargas/dimensões que mais são utilizadas nas empresas?

Critério de seleção de fontes:

Disponibilidade de consulta de artigos através via *web*;

Presença de mecanismos de busca através de palavras-chaves;

Garantia de resultados únicos através da busca de um mesmo conjunto de palavras-chaves;

Estar entre 2005 – 2015.

Palavras-chave:

Advantages, benefits, "stored procedures", load, BI, "Business Inteligence", DW, "Data Warehouse", procedure SQL, sp, problems, "business analytics" BI&A, "Big Data", disadvantage, advantages, benefits, ETL, XML, Kettle, Talend, Oracle Warehouse Builder, IBM Information Server, quality, "Business intelligence and analytics", "types of load", "dimension types", "types of historical data", "more used", business, industry.

Métodos de busca de fontes:

As fontes serão acessadas via *web*, portanto, no contexto desta revisão, não será considerada a busca manual.

Listagem de Fontes:

IEEE; ACM; DBLP; Elsevier; Thinkmind.

Critérios de Inclusão ou Exclusão dos artigos:

- Os artigos devem estar disponíveis na *web*;
- Os artigos encontrados devem apresentar textos completos dos estudos em formato eletrônico;
- Os artigos devem estar relacionados com as perguntas que foram feitas.

Buscas utilizadas:

- *(Advantages <or> benefits) <and> "stored procedures" <and> load <and> (BI <or> "Business Inteligence" <or> DW <or> "Data Warehouse");*
- *(Advantages <or> benefits) <and> ("procedure SQL" <or> "stored procedure" <or> sp) <and> load;*
- *Problems <and> ("Business intelligence <or> "business analytics" <or> "BI" <or> BI&A <or> "Big Data");*
- *(Problems <or> disadvantage) <and> (advantages <or> benefits) <and> ETL<and> XML;*
- *(Kettle <or> Talend <or> Oracle Warehouse Builder <or> IBM Information Server) <and> ETL;*
- *Problems <and> quality <and> ETL <and> (DW <or> "Data warehouse" <or> "Business intelligence and analytics" <or> "BI" <or> BI&A <or> "Big Data");*
- *(types of load <or> dimension types < types of historical data>) <and> "more used" (business <or> industry <or> BI <or> "Business Inteligence" <or> DW <or> "Data Warehouse").*

9.5 Apêndice V – Modelo de procedimento SQL gerado pela FGCod – atributos do tipo 1.

```
CREATE PROCEDURE [dw].[CARGA_CLIENTE] (@DATACARGA DATETIME)
AS
DECLARE
        @ALTEROU   CHAR(1), -- Variável para indicar se o registro na
Dimensao alterou ou não.
                                    -- S indica que alterou. N indica que
não houve alteração.
        @cod_cliente   INT,
        @data_nascimento   DATETIME,
        @cpf   VARCHAR(11),
        @nome_cliente   VARCHAR(50),
        @endereco   VARCHAR(50),
        @bairro   VARCHAR(20),
        @cidade   VARCHAR(20)


DECLARE Cursor_CARGA_CLIENTE CURSOR FOR
SELECT
        cod_cliente,
        data_nascimento,
        cpf,
        nome_cliente,
        endereco,
        bairro,
        cidade

FROM [staging].[TB_Aux_Cliente]
WHERE data_carga = @DATACARGA

BEGIN
  SET NOCOUNT ON
  BEGIN TRANSACTION


      OPEN Cursor_CARGA_CLIENTE
      FETCH Cursor_CARGA_CLIENTE INTO
                                          @cod_cliente,
                                          @data_nascimento,
```

```
                                    @cpf,
                                    @nome_cliente,
                                    @endereco,
                                    @bairro,
                                    @cidade


      WHILE(@@fetch_status=0)
      BEGIN
            SET @ALTEROU = 'S'
            IF EXISTS (SELECT 1 FROM [dw].[DIM_Cliente]
                        WHERE
                                    cod_cliente = @cod_cliente)
            BEGIN
                        SET @ALTEROU = 'N'

                        /*
=============================================================================
==
                        * Atualiza campos que possuem comportamento de
dimensão tipo 1 (sem histórico) */
                        UPDATE [dw].[DIM_Cliente]
                              SET
                                    data_nascimento =
@data_nascimento,
                                    cpf = @cpf,
                                    nome_cliente = @nome_cliente,
                                    endereco = @endereco,
                                    bairro = @bairro,
                                    cidade = @cidade
                              WHERE
                                    cod_cliente = @cod_cliente
            END

            -- Se não há registro ou não há alterações para DIM tipo 2, vai
inserir novo registro
            IF(@ALTEROU = 'S')
            BEGIN
                  INSERT INTO [dw].[DIM_Cliente] (
                              cod_cliente,
```

```
                              data_nascimento,
                              cpf,
                              nome_cliente,
                              endereco,
                              bairro,
                              cidade )
                        VALUES (
                              @cod_cliente,
                              @data_nascimento,
                              @cpf,
                              @nome_cliente,
                              @endereco,
                              @bairro,
                              @cidade )
                  END


            FETCH Cursor_CARGA_CLIENTE INTO
                                          @cod_cliente,
                                          @data_nascimento,
                                          @cpf,
                                          @nome_cliente,
                                          @endereco,
                                          @bairro,
                                          @cidade

            END

            COMMIT
            CLOSE Cursor_CARGA_CLIENTE
            DEALLOCATE Cursor_CARGA_CLIENTE
END
```

9.6 Apêndice VI – Modelo de procedimento SQL gerado pela FGCod – atributos do tipo 1 e 2.

```
CREATE PROCEDURE [dw].[CARGA_PRODUTO] (@DATACARGA DATETIME)
AS
DECLARE
        @ALTEROU   CHAR(1), -- Variável para indicar se o registro na
Dimensao alterou ou não.
                                    -- S indica que alterou. N indica que
não houve alteração.
        @codigo_produto   INT,
        @codigo_barra   INT,
        @valor   NUMERIC(10,2),
        @descricao   VARCHAR(25),
        @linha   VARCHAR(20)


DECLARE Cursor_CARGA_PRODUTO CURSOR FOR
SELECT
        codigo_produto,
        codigo_barra,
        valor,
        descricao,
        linha

FROM [staging].[TB_Aux_Produto]
WHERE data_carga = @DATACARGA

BEGIN
  SET NOCOUNT ON
  BEGIN TRANSACTION


      OPEN Cursor_CARGA_PRODUTO
      FETCH Cursor_CARGA_PRODUTO INTO
                                          @codigo_produto,
                                          @codigo_barra,
                                          @valor,
                                          @descricao,
                                          @linha
```

```
      WHILE(@@fetch_status=0)
      BEGIN
            SET @ALTEROU = 'S'
            IF EXISTS (SELECT 1 FROM [dw].[DIM_Produto]
                        WHERE
                                    codigo_produto = @codigo_produto)
            BEGIN
                        SET @ALTEROU = 'N'

                        /*
=============================================================================
==
                        * Atualiza campos que possuem comportamento de
dimensão tipo 1 (sem histórico) */
                        UPDATE [dw].[DIM_Produto]
                              SET
                                    codigo_barra = @codigo_barra
                              WHERE
                                    codigo_produto = @codigo_produto

                        /*
=============================================================================
==
                        *  Testando se houve alterações nos campos que
possuem comportamento Tipo 2 */
                        IF NOT EXISTS(SELECT 1 FROM [dw].[DIM_Produto]
                              WHERE
                                    codigo_produto = @codigo_produto
AND
                                    valor   =   @valor AND
                                    UPPER(descricao)   =
UPPER(@descricao) AND
                                    UPPER(linha)   =   UPPER(@linha)
AND
                                    fl_corrente = 'S' )
                              BEGIN
                                    UPDATE [dw].[DIM_Produto]
                                    SET
                                          fl_corrente   =   'N',
```

```
                                    dt_fim   =   @DATACARGA
                              WHERE
                                    codigo_produto = @codigo_produto
AND
                                    fl_corrente   =   'S';

                              SET @ALTEROU = 'S'
                        END

            END

            -- Se não há registro ou não há alterações para DIM tipo 2, vai
inserir novo registro
            IF(@ALTEROU = 'S')
            BEGIN
                  INSERT INTO [dw].[DIM_Produto] (
                              codigo_produto,
                              codigo_barra,
                              valor,
                              descricao,
                              linha,
                              dt_inicio,
                              dt_fim,
                              fl_corrente )
                  VALUES (
                              @codigo_produto,
                              @codigo_barra,
                              @valor,
                              @descricao,
                              @linha,
                              @DATACARGA,
                              NULL,
                              'S' )
            END


      FETCH Cursor_CARGA_PRODUTO INTO
                                    @codigo_produto,
                                    @codigo_barra,
                                    @valor,
                                    @descricao,
```

```
                                        @linha

        END

        COMMIT
        CLOSE Cursor_CARGA_PRODUTO
        DEALLOCATE Cursor_CARGA_PRODUTO
END
```